Vers des communications anonymes et efficaces

Gautier Berthou

UNIVERSITÉ DE GRENOBLE

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : **Informatique**

Arrêté ministériel : 7 août 2006

Présentée par

Gautier Berthou

Thèse dirigée par **Vivien Quéma**
et coencadrée par **Sonia Ben Mokhtar**

préparée au sein du **LIG**
et de **l'Ecole Doctorale de Mathématiques, Sciences et Technologies de l'Information, Informatique**

Vers des communications anonymes et efficaces

Thèse soutenue publiquement le **21 janvier 2014**,
devant le jury composé de :

Dr. Eddy Caron
ENS Lyon, Rapporteur
Prof. Pierre Sens
Université Pierre et Marie Curie - Paris VI, Rapporteur
Prof. Françoise Baude
Université de Nice Sophia-Antipolis, Examinatrice
Prof. Vivien Quéma
Grenoble INP, Directeur de thèse
Dr. Sonia Ben Mokhtar
CNRS, Co-Encadrante de thèse

Remerciements

Je tiens tout d'abord à remercier Professeur Françoise Baude, Professeur à l'Université de Nice Sophia-Antipolis, de me faire l'honneur de présider ce jury.

Je remercie également Docteur Eddy Caron, Maitre de conférences à l'ENS Lyon, et Professeur Pierre Sens, Professeur à l'Université Paris 6, d'avoir accepté d'être rapporteurs de cette thèse et d'avoir évalué mon travail de manière approfondie et constructive.

Je souhaite également remercier mes encadrants sans qui cette thèse n'aurait pas été possible. Tout d'abord, je remercie Professeur Vivien Quéma, Professeur à Grenoble INP, d'avoir accepté de me prendre en tant que doctorant et de m'avoir encadré tout au long de ces trois années. Je remercie ensuite, ma co-encadrante, Docteur Sonia Ben Mokhtar, Chargée de recherche au CNRS, pour son soutien et ses conseils.

Je tiens ensuite à remercier les personnes suivantes pour leurs apports divers et variés à ces trois années de thèse :

— Patrick Palmer, pour son encadrement de mes activités d'enseignements, ses conseils, ses retours et surtout son soutien dans les moments difficiles de ma thèse.
— L'ensemble de mes coauteurs, et notamment ce que je n'ai pas remerciés ci-dessus : Amadou Diarra, Ali Shoker et Alessio Pace.
— L'équipe enseignante de l'IUT 2 de Grenoble, avec qui j'ai eu la chance de travailler pendant mon activité d'enseignement.
— Tous les membres de la fanfare Pink it Black, pour tous les moments de joie, de détente, de défoulement et de bonheurs qu'ils m'ont apportés.
— Mes amis Grelakins, pour les moments de détentes que nous avons partagés ainsi que pour leur soutien et leur écoute dans les moments de doute.
— L'ensemble des doctorants des équipes Sardes et Erods, pour leur convivialité et leur bonne humeur. Avec une pensée particulière pour Baptiste et les débats plus ou moins constructifs que nous avons eu, ainsi que pour Pierre-louis qui ne cessera jamais de me surprendre.
— Les basketteurs du vendredi midi qui m'ont permis de garder la forme et d'évacuer mon trop plein d'énergie.
— La communauté européenne, pour avoir financé ma thèse, ainsi que le service public français pour l'ensemble de ma scolarité.

Pour finir je tiens à remercier ma famille pour tout ce qu'elle m'a apporté jusqu'à aujourd'hui et pour tout ce qu'elle m'apportera encore à l'avenir.

Préface

Cette thèse présente des recherches conduitent dans les équipes Sardes (Inria Grenoble – Rhône-Alpes / Laboratoire d'Informatique de Grenoble) et Erods (Laboratoire d'Informatique de Grenoble) dans le cadre de mon doctorat dans la spécialité "Informatique" de l'école doctorale "Mathématiques, Sciences et Technologies de l'Information, Informatique", au sein de l'Université de Grenoble. Ces recherches ont été menées sous la direction de Vivien Quéma (LIG/Grenoble INP) et Sonia Ben Mokhtar (LIRIS/CNRS).

Les travaux effectués pendant cette thèse ont porté sur la transmission d'informations dans les réseaux d'ordinateurs. Plus précisément sur deux sujets : (1) les communications anonymes sur Internet en présence de nœuds rationnels et (2) la diffusion à ordre uniformément total au sein d'une grappe de machines.

Chacun de ces travaux a donné lieu à une publication dans une conférence internationale :

RAC : a Freerider-resilient, Scalable, Anonymous Communication Protocol Sonia Ben Mokhtar, Gautier Berthou, Amadou Diarra, Vivien Quéma and Ali Shoker. *In proceedings of the 33rd International Conference on Distributed Computing Systems (ICDCS), July 2013.*

FastCast : a Throughput- and Latency-efficient Total Order Broadcast Protocol Gautier Berthou and Vivien Quéma. *In Procceding of Middleware 2013, December 2013.*

Table des matières

1

Introduction

Sommaire

La transmission d'informations sur les réseaux d'ordinateurs est source de nombreux problèmes : problèmes de sécurité, problèmes de montée en charge, problèmes de routage des messages, etc. L'importance et l'existence de ces problèmes dépendent de la nature et du type d'utilisation des réseaux : un réseau de petite taille, administré par une seule entité et comportant un faible nombre d'utilisateurs ne sera pas exposé aux mêmes problèmes de sécurité qu'un réseau de grande taille composé de multiples sous-réseaux administrés par différentes entités aux priorités divergentes. Il sera, par exemple, plus facile de trouver l'utilisateur mettant en danger la sécurité d'un réseau de petite taille que celle d'un réseau de grande taille. En contrepartie, les données circulant sur un réseau de petite taille sont souvent plus critiques et nécessitent plus de garanties de confinement et de sécurité du réseau. . Les travaux effectués dans le cadre de cette thèse portent sur deux problématiques : une problématique de sécurité dans le cadre d'Internet et une problématique de performances dans le cadre d'une grappe de machines. Ces deux problématiques prennent place dans deux types de réseaux d'ordinateurs dont les propriétés diffèrent fortement : Internet et un réseau local.

Dans ce chapitre, nous présentons tout d'abord le contexte des travaux de thèse. Nous introduisons ensuite nos deux contributions. Enfin, nous présentons l'organisation du document.

1.1 Contexte

1.1.1 Internet

Internet tire son nom du mot “interneting” signifiant interconnexion de réseau en anglais. Ce nom lui a été donné car c’est le plus grand réseau de réseaux existant. Internet constitue un réseau informatique mondial sans centre névralgique et composé de millions de réseaux aussi bien publics que privés, universitaires, commerciaux et gouvernementaux. Internet permet de transporter un large spectre d’informations et est le support de nombreuses applications. Le spectre de ces applications va de ce qui nous parait aujourd’hui le plus élémentaire, le courrier électronique, la messagerie instantanée et le World Wide Web, à des applications plus avancées telles que la voix sur IP, la télévision par Internet ou le streaming de musique.

Internet est constitué de réseaux interconnectés répartis dans le monde entier. Chaque réseau est rattaché à une entité (université, fournisseur d’accès à Internet, armée, etc.). Les réseaux sont interconnectés et peuvent échanger des informations crées en leur sein ou servir de relais entre deux autres réseaux. La figure 1.1 représente schématiquement le chemin suivit sur Internet par un message émis par un ordinateur A et reçu par un ordinateur B. Le message part de A et transite par le réseau R_1 auquel appartient A. Il passe ensuite par deux réseaux R_2 et R_3 servant de relais. Il atteint enfin le réseau R_4, auquel appartient B, et arrive à B.

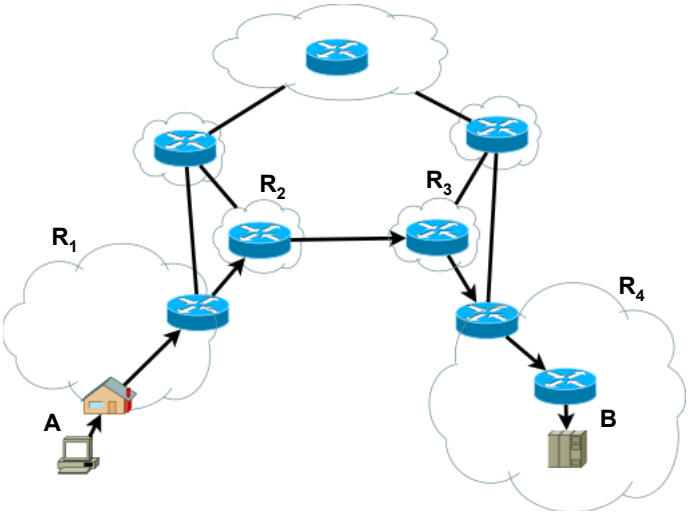


FIGURE 1.1 – Transmission d’un message sur Internet

La structure mondiale et non centralisée d’Internet est source de nombreuses difficultés, l’une des plus importantes étant la question de la gouvernance de l’Internet. Telle que définie par le groupe de travail sur la gouvernance d’Internet, cette gouvernance est “l’élaboration et l’application par les États, le secteur privé et la société civile, dans le cadre de leurs rôles respectifs, de principes, normes, règles, procédures de prise de décisions et programmes communs propres à modeler l’évolution et l’usage de l’Internet”. Comme le montre cette définition, la gouvernance d’Internet est à l’image du réseau : distribuée. Il existe ainsi de nombreuses organisations visant à gérer ou

promouvoir différents aspects du réseau et dont les prérogatives peuvent parfois se chevaucher et les objectifs diverger. Ces divergences d'objectifs et de points de vue se font notamment ressentir au niveau des états. Chaque état cherche à contrôler la partie du réseau se trouvant sur son territoire. Il en résulte qu'il est très difficile pour un utilisateur de savoir quelle législation s'applique aux données qu'il envoie sur Internet. Ainsi, un utilisateur français, dont certaines informations personnelles sont protégées par la loi "Informatique et Liberté", peut voir ses informations révélées si elles sont envoyées sur une partie du réseau ne se trouvant pas en France. Si l'on prend l'exemple de la figure 1.1, il est possible que A et B soient dans le même pays et que leurs réseaux respectifs (R_1 et R_4) soient soumis aux lois de ce pays, tandis que les réseaux relais (R_2 et R_3) se trouvent dans d'autres pays ayant des législations différentes. A et B pourraient alors penser communiquer sans être surveillés alors que le pays où se trouve R_4 surveille tout ce qui passe sur son réseau. Il est, par exemple, de notoriété publique que l'état chinois surveille toutes les informations circulant sur son territoire à l'aide du grand firewall [1]. Des révélations récentes ont montré que de nombreux états, dont les Etats Unis [2], s'adonnaient au même genre de pratiques. Nous voyons donc que la question de la sécurité et de la confidentialité de l'information circulant sur Internet est une problématique complexe qui ne peut être garantie par les états et qui est centrale si l'on veut assurer la liberté d'expression des utilisateurs d'Internet.

1.1.2 Réseau local

A l'opposé d'Internet, un réseau local est composé d'un unique réseau administré par une entité bien définie. Les problèmes de gouvernance et de sécurité se posent donc moins. Un réseau local est un réseau de taille réduite dans lequel les machines sont interconnectées à l'aide de switchs ou de routeurs et communiquent directement les unes avec les autres sans passer par Internet. Ce type de réseau peut par exemple se trouver dans les universités, au sein des administrations ou dans les entreprises. Un réseau local peut disposer d'un point d'accès vers Internet. Une organisation de réseau utilisée par de nombreux services Internet consiste à connecter une machine à Internet d'un côté et à un réseau local hébergeant d'autres machines de l'autre. Cette configuration est utilisée de la façon suivante : la machine connectée à Internet reçoit les requêtes des utilisateurs et y répond à l'aide de services ou d'informations obtenus en communiquant avec les machines connectées au réseau local. Un exemple typique d'utilisation de cette organisation est l'architecture trois-tiers. Celle-ci consiste à avoir une première machine connecté à Internet et chargée du dialogue avec les utilisateurs, une seconde machine chargée du traitement des requêtes et une troisième machine chargée de la base de données. Le schéma de communication est alors le suivant : les utilisateurs communiquent avec la première machine par l'intermédiaire d'Internet, la première machine transmet les requêtes à la seconde machine par l'intermédiaire du réseau local et la seconde machine communique avec la troisième par l'intermédiaire du réseau local, pour récupérer ou stocker des informations dans la base de données. Cette architecture permet, par exemple, d'assurer un certain niveau de performance en répartissant les tâches provenant d'Internet sur différentes machines, elle permet aussi

1. Grand firewall de chine : http ://fr.wikipedia. org/wiki/Grand_Firewall_de_Chine
2. Prism : http ://fr.wikipedia.org/wiki/PRISM_%28programme_de_surveillance%29

d'assurer un certain niveau de sécurité (un utilisateur malveillant sera bloqué au niveau de la machine connectée à Internet).

Dès lors que le réseau local sert d'infrastructure sous-jacente à des services informatiques, la question de la performance se pose. Il faut en effet que les communications sur le réseau local soient assez rapides pour que les utilisateurs aient une bonne qualité de service. Pour ce faire il faut que les communications sur le réseau local aient une *latence* faible, c'est-à-dire dire que le temps entre l'envoi d'un message et sa réception soit faible lorsque le réseau n'est pas saturé. Il faut aussi que les communications aient un *débit* élevé, c'est-à-dire que les machines livrent un grand nombre de messages par seconde lorsque le réseau est saturé. Au niveau matériel, une façon d'assurer de bonnes performances est d'organiser le réseau en grappe de machines homogènes. c'est-à-dire, de réunir géographiquement les machines, ce qui permet d'éviter la latence due à la circulation de l'information sur les câbles réseau, et de les connecter entre elles par un switch, ce qui permet une communication rapide entre les machines tout en permettant de répartir la charge réseau. Contrairement à Internet, dont la structure est difficile à définir, les réseaux locaux ont une structure bien définie et connue à l'avance. Cette caractéristique, couplée à la connaissance du type d'informations et des schémas de communications présents sur le réseau, permet d'optimiser les protocoles de communications afin d'assurer une latence faible et un débit élevé aux communications. Ces optimisations sont indispensables pour pouvoir répondre aux attentes des utilisateurs de services informatiques s'appuyant sur des réseaux locaux.

1.2 Contribution

Afin de répondre aux deux challenges présentés ci-dessus (la confidentialité de l'information circulant sur Internet, et la performance des protocoles de communication dans le cadre d'un réseau local) nous avons proposé deux contributions. La première contribution porte sur l'anonymat des communications et la seconde sur la diffusion à ordre uniformément total.

1.2.1 Anonymat des communications

Lorsqu'un utilisateur d'Internet effectue une communication il peut chiffrer celle-ci de sorte qu'il soit impossible de connaitre le contenu de cette communication. Cependant, une personne interceptant cette communication est capable de savoir qui communique avec qui. En effet, les protocoles de transport de messages sur Internet (TCP et UDP) imposent à chaque message de contenir les adresses de son expéditeur et de son destinataire. Ceci est une limitation qui peut se révéler problématique. Imaginons que la NSA intercepte une communication chiffrée entre un lanceur d'alertes et wikileaks. Puis, que quelques semaines plus tard des informations sensibles soient publiées sur wikileaks. La NSA aura alors de forts soupçons sur l'origine des informations et rendra une "visite de courtoisie" au lanceur d'alertes. C'est pour palier à ce problème que les protocoles de communications anonymes ont été inventés. Le but de ces protocoles et de permettre à chaque utilisateur de communiquer avec d'autres utilisateurs sans qu'il soit possible de savoir qui envoie ou reçoit des messages au sein de la masses des utilisateurs. De cette façon la NSA pourra savoir que wikileaks et

le lanceur d'alertes ont participé à un protocole de communications anonymes, mais des milliers d'autres personnes auront aussi participé à ce même protocole et il sera impossible de faire un lien direct entre le lanceur d'alertes et wikileaks (en effet, le lanceur d'alertes peut avoir participé au protocole sans rien envoyer).

Nous avons passé en revue les protocoles de communications anonymes [1, 2, 3, 4, 5, 6] et nous avons constaté qu'il n'existait pas de protocole ne s'appuyant pas sur un nombre limité de serveurs, capable de monter en charge et fonctionnant en présence de nœuds rationnels. Ces trois propriétés sont importantes. La première propriété, ne pas s'appuyer sur un nombre limité de serveurs, est importante car plus le nombre de serveurs fournissant l'anonymat est petit, plus il est facile de les attaquer pour casser l'anonymat des communications. De plus, un nombre réduit de serveurs implique une montée en charge limitée. En effet, le même nombre de serveurs doit soutenir la charge demandée par les clients quel que soit le nombre de clients, or les capacités des serveurs sont limitées. La capacité de monter en charge est importante car plus le nombre d'utilisateurs est important, plus il est difficile de lier une communication à un utilisateur : s'il y a cent utilisateurs et que l'un d'entre eux envoie un message à wikileaks, la NSA n'a qu'à rendre cent visites de courtoisies, alors qu'elle devra en rendre 10000 s'il y a 10000 utilisateurs. Pour finir, il est connu que les systèmes ne s'appuyant pas sur des serveurs (les systèmes pair-à-pair) souffrent de vulnérabilité face aux comportements dit rationnels [7, 8, 9, 10]. Ces comportements consistent à dévier du protocole pour optimiser le bénéfice tiré de son utilisation. La déviation peut se faire en ne participant pas à certaines étapes du protocole. De cette façon l'utilisateur peut économiser des ressources telles que ça bande passante, sa mémoire, sa puissance de calcul, etc. Les utilisateurs présentant ce comportement sont qualifiés de "freeriders" [7, 11]. La déviation peut aussi se faire en effectuant des actions n'étant pas prévus par le protocole. Ceci a un coût en ressources, mais peut permettre de dévier le comportement du système dans une direction plus favorable à l'utilisateur rationnel. Ce comportement, plus large que le comportement des freeriders, est étudié par les travaux sur les nœuds rationnels [8, 9, 10]. Si ce genre de comportement peut représenter un gain pour l'utilisateur rationnel, il est nuisible pour la communauté des utilisateurs dans son ensemble. En effet, si un grand nombre d'utilisateurs a un comportement rationnel, toute la charge de travail est soutenue par les utilisateurs qui exécutent correctement le protocole. Ce phénomène est appelé tragédie du commun [12] et peut avoir de graves conséquences en termes de performance (la bande passante des utilisateurs suivant le protocole n'est pas extensive) et en termes de sécurité (il suffit d'attaquer les utilisateurs suivant le protocole pour casser l'anonymat).

Nous avons donc proposé *RAC*, le premier protocole de communications anonymes fonctionnant sur un modèle pair-à-pair en présence de nœuds rationnels et capable de monter en charge. Nous avons simulé *RAC* à l'aide de Omnet++ [13] et comparé ses performances aux seuls protocoles de l'état de l'art capables de fonctionner en présence de nœuds rationnels : Dissent v1 [4] et Dissent v2 [5]. Notre évaluation montre que *RAC* assure un bien meilleur débit que Dissent v1 et v2. Elle montre aussi une montée en charge efficace de *RAC*, là où le débit des deux versions de Dissent décroit avec le nombre de nœuds dans le système. Nous avons ensuite évalué mathématiquement l'anonymat garanti par *RAC*, *Onion routing* [1] ainsi que les deux versions de Dissent. Cette évaluation montre que si l'anonymat garanti par *RAC* est inférieur à celui garanti par Dissent, il est supérieur à celui garanti par *Onion routing*. Hors, ce dernier est le

mécanisme d'anonymat le plus utilisé aujourd'hui, notamment grâce à Tor [2].

1.2.2 Diffusion à ordre uniformément total

L'une des utilisations de réseaux locaux sous-jacente à un service informatique est la réplication de machines à états [14]. La réplication de machines à états consiste à faire effectuer les mêmes opérations par plusieurs machines afin de garantir qu'un service sera toujours accessible, même si une ou plusieurs machines viennent à cesser de fonctionner. Si toutes les machines partent d'un même état initial et effectuent les mêmes opérations dans le même ordre elles seront toutes en permanence dans le même état. De cette façon, si une machine s'arrête, une autre pourra prendre sa place sans qu'il y ait de perte d'informations lors de la prise de relais. La réplication de machines à états s'appuie sur deux techniques pour assurer que toutes les machines effectuent les mêmes opérations dans le même ordre : (1) les machines exécutent toutes le même programme séquentiel et (2) les requêtes sont envoyées aux machines à l'aide d'un protocole de diffusion à ordre uniformément total. Un protocole de diffusion à ordre uniformément total permet d'envoyer des messages à de multiples machines en assurant (1) que si une machine livre un message, toutes les machines correctes livreront ce message ; (2) que toutes les machines livrent les messages dans le même ordre : si une machine livre un message m_1 avant un message m_2 alors toutes les machines livreront m_1 avant m_2. Le premier point garantit que toutes les machines correctes effectueront les mêmes opérations. Le second point assure que toutes les requêtes seront livrées dans le même ordre par toutes les machines. Comme les machines exécutent un programme séquentiel, garantir que toutes les machines livrent les requêtes dans le même ordre garantit que toutes les machines effectueront les mêmes opérations dans le même ordre.

De nombreux travaux ont été effectués afin d'optimiser les performances des protocoles de diffusion à ordre uniformément total. Un grand nombre de ces optimisations porte sur la latence [15], c'est-à-dire le temps entre l'envoi d'une requête et le moment où elle a été livrée par toutes les machines, lorsque le système n'est pas utilisé aux limites de ses capacités. Optimiser la latence est nécessaire car cela assure à l'utilisateur un temps de réponse faible lorsque le système n'est pas utilisé au maximum de ses capacités. D'autres travaux ont été effectués afin d'optimiser le débit des protocoles de diffusion à ordre uniformément total [16]. Le débit correspond au nombre de messages livrés par seconde lorsque le système est utilisé au maximum de ses capacités. Cette optimisation est nécessaire car elle permet au système de traiter un grand nombre de requêtes en période de pointe. Malgré le grand nombre de travaux traitant de la diffusion à ordre uniformément total, il n'existe pas de protocole permettant d'avoir à la fois une latence faible et un débit optimal. Il faut donc faire un choix entre débit et latence. Choix d'autant plus complexe qu'il est difficile, voire impossible, de garantir qu'un système ne sera jamais utilisé au maximum de ses capacité ou, à l'inverse, qu'il sera toujours utilisé au maximum de ses capacités.

Afin de résoudre ce problème, nous avons proposé *FastCast*, le premier protocole de diffusion à ordre uniformément total assurant à la fois un débit optimal et une latence faible. Afin d'assurer une latence faible, *FastCast* s'appuie sur la multi-diffusion IP. Ce mécanisme permet de faire parvenir un même message à de multiples destinataires en un seul envoi. Il a cependant l'inconvénient de perdre des messages lorsque le réseau est saturé. Afin d'éviter ces pertes, *FastCast* prévient la perte de messages en

régulant le débit d'envoi de chaque machine. La nouveauté de *FastCast* réside dans le sous protocole, exécuté par toutes les machines, qui calcule de façon dynamique le débit auquel chaque machine peut envoyer des messages. Nous avons implémenté *FastCast* en C++ afin de comparer ses performances aux performances garanties par deux protocoles récents et constituant l'état de l'art : LCR [16] et Ring Paxos [17]. Le premier garantit un débit optimal tandis que le second tente de garantir à la fois un débit élevé et une latence faible. Notre évaluation, effectuée dans une grappe de huit machines, montre que *FastCast* garantit un débit optimal et une latence faible. Plus précisément, *FastCast* garantit un débit jusqu'à 86% plus élevé que Ring Paxos et une latence jusqu'à 247% plus faible que LCR.

1.3 Organisation de ce document

Ce document est structuré en deux parties, chacune divisée en six chapitres. Chaque partie présente le travail effectué sur l'une des deux problématiques étudiées : la première partie traite du problème des communications anonymes en présence de nœuds rationnels, tandis que la deuxième partie traite des performances des mécanismes de diffusions au sein d'une grappe de machines. Ces deux parties, dont la décomposition est présentée ci-dessous, sont suivies d'une conclusion générale résumant nos travaux et présentant des perspectives de travaux futurs.

1.3.1 Première partie : Communications anonymes en présence de machines rationnelles

La première partie se compose comme suit. Le premier chapitre sert d'introduction ; il présente ce que sont les communications anonymes, le cadre dans lequel elles ont lieu et le type d'adversaires auxquels elles doivent résister.

Le second chapitre présente l'état de l'art en matière de communications anonymes. Il présente les deux grandes familles de protocoles de communications anonymes que sont DCNet et *Onion routing* et détaille, pour chacune, les forces et les faiblesses des protocoles la composant. Il finit en expliquant pourquoi un nouveau protocole est nécessaire.

Le troisième chapitre présente notre solution au problème des communications anonymes en présence de nœuds rationnels : le protocole *RAC*. Ce chapitre commence par présenter les idées clés nous permettant de proposer un protocole de communications anonymes fonctionnant en présence de nœuds rationnels tout en étant capable de monter en charge, avant de présenter les détails du protocole.

Les quatrième et cinquième chapitres prouvent et évaluent les performances de *RAC*. Dans le quatrième chapitre il est d'abord prouvé que *RAC* assure l'anonymat des communications, puis il est prouvé que le protocole garantit un équilibre de Nash et garantit donc que les nœuds rationnels suivront le protocole à la lettre. Le cinquième chapitre évalue les performances du protocole à l'aide de simulations et de formules mathématiques.

Pour finir, le sixième chapitre sert de conclusion à cette partie.

1.3.2 Deuxième partie : Diffusion à ordre uniformément total

La deuxième partie se compose comme suit. Le premier chapitre sert d'introduction et présente les deux métriques sur lesquelles se sont portées les recherches dans le domaine de la diffusion à ordre uniformément total : la latence et le débit. Il explique notamment pourquoi garantir une bonne latence ne garantit pas forcément un bon débit.

Le deuxième chapitre présente les différentes architectures permettant de construire un protocole de diffusion à ordre uniformément total. Les limitations de chacune de ces architectures sont passées en revue afin d'expliquer pourquoi il n'existe pas aujourd'hui de protocole permettant d'assurer à la fois une bonne latence et un débit optimal.

Le troisième chapitre présente *FastCast*, le premier protocole permettant d'assurer une diffusion à ordre uniformément total garantissant à la fois un débit optimal et une latence faible. Ce chapitre présente les trois sous-protocoles constituant *FastCast*, en s'attardant tout particulièrement sur le plus important d'entre eux : le protocole d'allocation de bande passante.

Le quatrième chapitre prouve différents aspects de *FastCast*. Il prouve tout d'abord que *FastCast* est bien un protocole de diffusion à ordre uniformément total. Il prouve pour ce faire que *FastCast* assure l'intégrité, l'accord uniforme et l'ordre total des envois de message. Il prouve ensuite que le sous-protocole d'allocation de bande passante permet bien d'assurer que la bande passante utilisée ne sera jamais supérieure à la bande passante que le réseau peut transmettre. Il prouve enfin que le sous-protocole d'allocation de bande passante effectue les allocations de sorte que la bande passante utilisée soit égale à la bande passante optimale, lorsque c'est possible.

Le cinquième chapitre évalue les performances de *FastCast* et les compare à deux protocoles représentant l'état de l'art : Ring Paxos pour sa latence faible, et LCR pour son débit optimal. Il est alors montré que *FastCast* surclasse ces deux protocoles.

Pour finir le sixième chapitre sert de conclusion à cette partie.

Première partie

Communications anonymes en présence de machines rationnelles

2

Introduction

Sommaire

Internet est devenu l'un des principaux moyens de dissémination d'informations. Comme illustré par le printemps arabe [1], c'est même parfois le seul moyen de transmettre de l'information. Cependant, de par la structure même des protocoles sous-jacents à Internet, Internet peut aussi être utilisé pour amasser des informations, parfois sensibles, à propos des personnes qui l'utilisent. Ceci a notamment été illustré lors des récentes révélations sur le projet PRISM [2] : la NSA aurait la capacité de surveiller 75% du trafic Internet aux Etats Unis [18]. C'est pourquoi de nombreux protocoles [1,2,3,4,5,6] ont été proposés pour assurer l'anonymat des communications sur Internet. Cependant, comme nous allons le montrer dans le chapitre 3, aucun de ces protocoles n'est pleinement satisfaisant. Avant de passer en revue les protocoles de communications anonymes, il nous faut expliquer ce qu'est une communication anonyme et ce que l'on attend d'un protocole de communications anonymes. C'est l'objet de ce chapitre.

2.1 Qu'attend-t-on d'un protocole de communications anonymes ?

Avant de détailler les protocoles de communications anonymes existants il faut comprendre ce qu'est une communication anonyme et définir ce que l'on attend aujourd'hui

1. Printemps arabe : http ://fr.wikipedia.org/wiki/Printemps_arabe
2. Prism : http ://fr.wikipedia.org/wiki/PRISM_%28programme_de_surveillance%29

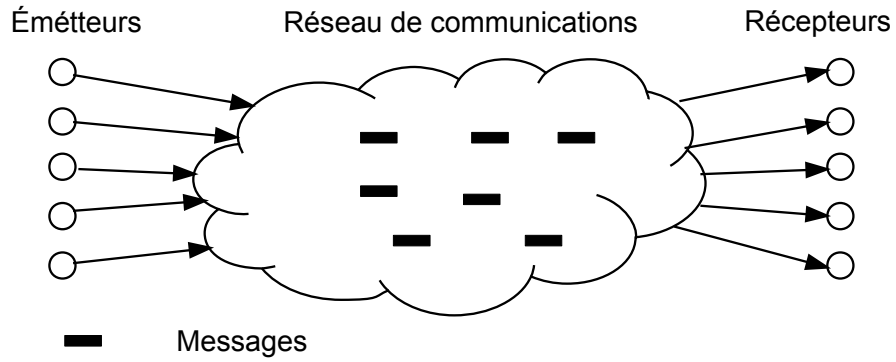


FIGURE 2.1 – Modélisation du système

d'un protocole de communications anonymes. Pour ce faire nous commencerons par définir le cadre dans lequel un système de communications anonymes est déployé. Nous définirons ensuite ce qu'est l'anonymat en termes de communications sur Internet. Pour cela nous nous appuyons sur la terminologie définie par Pfitzmann et Hansen [19]. Les définitions de cette terminologie étant faites du point de vue d'un adversaire tentant de casser l'anonymat des communications nous définirons, en section 2.1.2, les capacités de l'adversaire, avant de définir, en section 2.1.3, les différents types d'anonymats. Nous conclurons en section 2.1.4 en listant les caractéristiques indispensables à un bon système de communications anonymes.

2.1.1 Cadre

Le système que nous étudions est composé d'un groupe d'acteurs, que nous appellerons nœuds, qui peuvent jouer différents rôles : client, serveur, pair d'un réseau pair-à-pair... Ces acteurs sont connectés entre eux par un réseau de communication. Tout nœud peut communiquer avec tout autre nœud par l'intermédiaire de ce réseau. Nous appellerons lien le chemin de communication entre deux nœuds. Comme illustré dans la figure 2.1, la communication entre les nœuds se fait par l'échange de messages entre un nœud émetteur et un nœud récepteur. Par la suite, nous appellerons communication tout échange d'un message ou d'un ensemble de messages entre deux nœuds. Notez qu'un nœud peut être à la fois un émetteur et un récepteur et qu'il peut changer de rôle au cours du temps.

2.1.2 Adversaire

Dans le cadre défini ci-dessus, un adversaire est une entité qui cherche à surveiller les communications ayant lieu dans le système. Il peut par exemple chercher à savoir qui a envoyé ou reçu un message donné, quels sont les messages envoyés par un nœud donné ou quels nœuds communiquent avec un nœud donné. Les caractéristiques de cet adversaire sont décrites par quatre caractéristiques définies par J.-F. Raymond [20] :

Capacité. Un adversaire peut être passif ou actif. Un adversaire passif est un adversaire capable d'enregistrer le trafic sur les liens réseau entrant et sortant des nœuds du système. En plus du trafic brut, l'adversaire passif peut enregistrer des

méta-informations telles que la taille des messages et leurs heures d'arrivée et de départ. Un adversaire actif a les mêmes capacités qu'un adversaire passif, auxquelles il ajoute la capacité de manipuler le trafic circulant sur le réseau : il peut supprimer ou modifier des messages circulant sur le réseau ; il peut aussi injecter ses propres messages dans le réseau ; il peut renvoyer des messages qu'il a enregistré par le passé. Ces capacités peuvent être obtenues en contrôlant des liens du réseau de communication ou en contrôlant des nœuds participant au système d'anonymat.

Visibilité. La visibilité d'un adversaire définit la portion du réseau qu'un adversaire peut observer de façon passive ou active. Un adversaire global est un adversaire qui a la capacité d'observer l'ensemble du réseau de communication. Un adversaire n'ayant pas cette capacité est dit partiel. Un adversaire partiel ne peut, par exemple, observer que le trafic entrant ou sortant des nœuds qu'il contrôle. Il peut par exemple s'agir d'un fournisseur d'accès à Internet qui ne peut contrôler le trafic que sur son réseau et n'a aucune information sur ce qui se passe sur le réseau des autres fournisseurs d'accès. L'adversaire le plus souvent choisi lors de la conception d'un protocole de communications anonymes est un adversaire global, car un protocole capable de résister à un adversaire global peut aussi résister à un adversaire partiel.

Mobilité. Un adversaire qui n'a pas les infrastructures pour être global peut avoir la capacité de modifier la sous-partie du réseau qu'il surveille. Cet adversaire est alors dit adaptatif. Par exemple, la justice d'un état peut surveiller le trafic entrant et sortant d'une compagnie ou d'un utilisateur, puis, en fonction des informations récoltées, décider de mettre sur écoute d'autres machines. Un adversaire n'ayant pas cette capacité d'adaptation est dit statique.

Participation. Un adversaire peut être interne ou externe. Un adversaire interne est un adversaire qui contrôle des nœuds participant au réseau d'anonymat. Un adversaire interne peut être passif ; il se contente alors d'enregistrer le trafic entrant et sortant des nœuds qu'il contrôle tout en suivant normalement le protocole. Il peut aussi être actif et essayer d'obtenir plus d'informations en faisant dévier du protocole les nœuds qu'il contrôle. Un adversaire externe ne contrôle pas de nœud ; il n'a d'influence que sur le réseau de communication.

Le type d'adversaire le plus souvent utilisé lors de la conception de protocoles d'anonymat est un adversaire passif et global. Cet adversaire a l'avantage d'être puissant. Un protocole assurant de l'anonymat en présence d'un tel adversaire garantira de l'anonymat en présence d'adversaires ayant accès à moins d'informations, tel qu'un adversaire passif, partiel et adaptatif. Il est parfois avancé que le modèle d'un opposant passif et global est irréaliste pour de larges systèmes d'anonymat déployés sur Internet [21]. Cet irréalisme est justifié par l'aspect décentralisé d'Internet qui obligerait de nombreuses juridictions à coopérer afin de créer un tel adversaire. Cependant, le grand firewall de Chine [3] ou le patriot act [4] et ses dérivés [5] peuvent mettre en doute l'irréalisme de ce type d'adversaire. Il est aussi possible de reprocher à ce modèle d'adversaire de ne pas aller assez loin. En effet, si une organisation a les moyens de créer un adversaire global

3. Grand firewall de chine : http ://fr.wikipedia.org/wiki/Grand_Firewall_de_Chine
4. Usa patriot act : http ://fr.wikipedia.org/wiki/Patriot_Act
5. Prism : http ://fr.wikipedia.org/wiki/PRISM_%28programme_de_surveillance%29

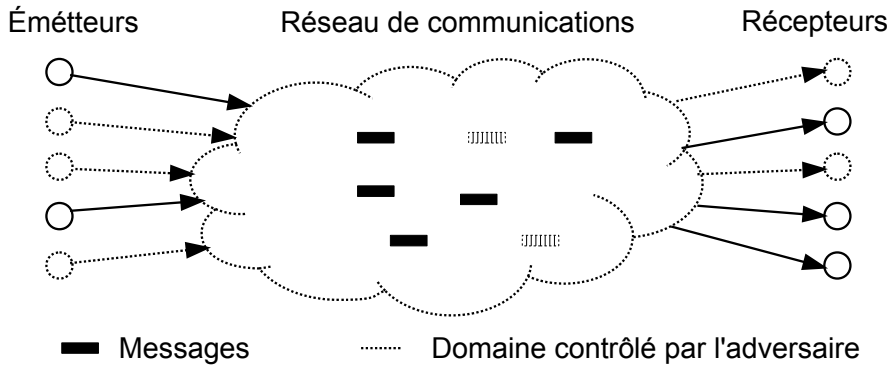


FIGURE 2.2 – Adversaire global et actif

et passif, il est difficile d'imaginer qu'elle ne puisse pas manipuler les données circulant sur le réseau. Nous avons donc choisi pour notre étude de considérer un adversaire global, actif et interne. La figure 2.2 récapitule les parties du système sur lesquels un tel adversaire a de l'influence.

2.1.3 Type et force d'anonymat

Dans l'idéal, un nœud utilisant un protocole de communications anonymes souhaiterait qu'il soit impossible de le lier à toute communication, à quelque niveau que ce soit. Cependant, il est impossible d'empêcher un adversaire assez puissant de lister l'ensemble des nœuds participant au protocole de communications anonymes. Un nœud utilisant un protocole de communications anonymes ne peut donc se dissimuler que parmi les nœuds participant à ce protocole. C'est pourquoi Pfitzmann et Hansen [19] ont défini l'anonymat comme "le fait de ne pas être identifiable au sein d'un groupe de nœuds, appelé groupe d'anonymat". Si l'on considère, par exemple, un nœud ayant émis un message, son groupe d'anonymat est l'ensemble des nœuds qui peuvent avoir envoyé le message. Sans protocole de communications anonymes, cet ensemble est réduit au seul nœud ayant envoyé le message. Avec un protocole de communications anonymes assurant l'anonymat des émetteurs, il existera un nombre N de nœuds ayant, du point de vue de l'adversaire, la même probabilité $1/N$ d'être émetteur de ce message. Réciproquement, un protocole de communications anonymes assurant l'anonymat des destinataires assurera que pour tout message il existe un nombre N de nœuds ayant, du point de vue de l'adversaire, la même probabilité $1/N$ d'être destinataires de ce message.

On constate, intuitivement, que plus N est grand, plus on pourra considérer que l'anonymat est fort. Si un message de dissidence est envoyé à l'aide d'un protocole de communication assurant un groupe d'anonymat de taille deux, le gouvernement n'aura qu'à enquêter sur les deux membres du groupe pour trouver l'émetteur du message. Alors que, si le groupe d'anonymat est de taille mille, le gouvernement aura beaucoup plus de mal à trouver l'émetteur de message parmi ces mille nœuds. Il apparait alors qu'un protocole de communications anonymes idéal devrait assurer pour chaque communication un groupe d'anonymat de taille égale au nombre de nœuds participant

au système. Cependant, il ne faut pas oublier que dans notre cas l'adversaire est interne. Cela, signifie que l'adversaire contrôle certains nœuds participant au système, il peut donc savoir si ces nœuds ont participé à une communication donnée. Le groupe d'anonymat idéal est donc de taille $N - A$ avec N le nombre de nœuds dans le système et A le nombre de nœuds contrôlés par l'adversaire.

L'adversaire que nous prenons en compte dans cette étude est un adversaire actif, global et interne. Il se peut donc qu'il contrôle le nœud destinataire d'un message. Un protocole assurant *l'anonymat des émetteurs* doit donc garantir que l'émetteur d'un message fait partie d'un groupe d'anonymat non seulement pour un adversaire observant le réseau, mais aussi pour le destinataire du message. Il est à noter que les protocoles de communications anonymes ne concernent que la transmission des messages, ils ne s'intéressent pas à leur contenu. Ainsi, si un utilisateur envoie un message contenant son identité, son anonymat ne sera pas assuré si le destinataire du message est un nœud contrôlé par l'adversaire.

De la même façon qu'un protocole assurant l'anonymat de l'émetteur ne permet pas au destinataire d'un message de connaitre l'identité de son émetteur, un protocole assurant *l'anonymat des destinataires* doit garantir que l'émetteur d'un message ne peut connaitre l'identité du destinataire de ce message. Pour qu'un nœud puisse envoyer un message à un destinataire sans en connaitre l'identité, les protocoles d'anonymat utilisent des pseudonymes ou des mécanismes équivalents. Ces pseudonymes garantissent que les actions d'envoyer ou de recevoir un message sont liées à un identificateur différent de la vraie identité des nœuds.

En plus de l'anonymat de l'émetteur et de l'anonymat du destinataire Pfitzmann et Hansen [19] définissent un dernier type d'anonymat : *l'anonymat des relations*. Cet anonymat assure qu'un adversaire sélectionnant deux nœuds dans le système ne peut déterminer si ces nœuds communiquent entre eux. Cet anonymat est plus faible que les deux anonymats précédemment décrits. En effet, un protocole assurant l'anonymat des émetteurs (respectivement des destinataires) garantit l'anonymat des relations, car il assure que pour toute communication il est impossible pour l'adversaire de savoir qui émet (respectivement réceptionne) la communication, il est donc impossible pour l'adversaire de savoir qui communique avec qui. À l'opposé, comme nous le verrons dans le chapitre 3, il existe des protocoles qui assurent l'anonymat des relations sans assurer l'anonymat des émetteurs et l'anonymat des destinataires : l'adversaire est capable de savoir quel nœud a envoyé quel message et de savoir quel nœud a reçu quel message, mais l'adversaire ne peut faire le lien entre les messages envoyés et les messages reçus (car ceux-ci sont modifiés durant les transmissions successives) et ne peut donc faire le lien entre les émetteurs et les destinataires.

2.1.4 Récapitulatif

Si les premiers protocoles de communications anonymes n'assuraient que l'anonymat des relations, il est aujourd'hui attendu de tout protocole de communications anonymes qu'il assure l'ensemble des trois anonymats présentés ci-dessus :

Anonymat des relations Un protocole ne garantissant pas au moins l'anonymat des relations ne peut revendiquer être un protocole de communications anonymes.

Anonymat des émetteurs C'est une garantie importante car un utilisateur de système de communications anonymes veut non seulement être sûr qu'un observateur

extérieur ne sera pas capable de savoir avec qui il communique, mais il veut aussi être sûr de ne pas se faire piéger par un destinataire contrôlé par l'adversaire.

Anonymat des destinataires Cette garantie est indispensable car elle permet d'avoir un échange bidirectionnel. L'émetteur d'une communication anonyme garantissant l'anonymat de l'émetteur veut souvent pouvoir recevoir une réponse ou être recontacté par le destinataire de cette communication.

Les protocoles de communications anonymes doivent de plus répondre à d'autres impératifs non directement liés à l'anonymat des communications. Comme tout protocole de communications, un protocole de communications anonymes doit être capable de monter en charge en termes de nombre de nœuds dans le système. Cette capacité à monter en charge est d'autant plus importante que plus le nombre de nœuds dans le système est grand plus les groupes d'anonymat peuvent être grands et plus l'anonymat peut être fort.

Comme nous allons le voir en chapitre 3, une solution pour construire des protocoles capables de monter en charge et pour assurer une meilleur résistance aux attaques est de construire des protocoles pair-à-pair. Les protocoles pair-à-pair ont l'avantage de distribuer leur charge sur l'ensemble des nœuds du système. Ils évitent ainsi les goulots d'étranglements qui peuvent se former au niveau des serveurs et empêcher la montée en charge. Cette avantage est cependant contrebalancé par l'existence de nœuds dit "rationnels" qui peuvent anéantir les performances du système. Les nœuds rationnels sont des nœuds qui dévient du protocole initial afin d'optimiser le bénéfice qu'ils tirent du système. La déviation peut se faire en ne participant pas à certaines étapes du protocole, dans le but d'éviter de partager des ressources (bande passante, puissance de calcul, mémoire, etc.). C'est le comportement observé chez les "freeriders" [7, 11]. Un nœud peut aussi dévier du protocole en exécutant des actions n'étant pas prévu par le protocole. Ceci a un coût en ressources, mais peut permettre de dévier le comportement du système dans une direction plus favorable au nœud rationnel. Ce comportement, plus large que le comportement des freeriders, est étudié par les travaux sur les nœuds rationnels [8, 9, 10]. Les nœuds rationnels peuvent entrainer un phénomène de tragédie du commun [12] : au-delà d'une certaine masse critique, le nombre de nœuds rationnels peut avoir un tel impact sur le protocole que celui-ci ne fonctionne plus. Un protocole de communications anonymes pair-à-pair se doit d'être résistant à ce genre de comportement.

2.2 Contribution

Nous avons conçu *RAC*, le premier protocole de communications anonymes permettant d'assurer les trois types d'anonymat (anonymat des relations, anonymat des émetteurs, anonymat des destinataires) tout en permettant de monter en charge et en gérant les nœuds rationnels. Afin de pouvoir monter en charge tout en assurant de bonnes performances, *RAC* s'appuie sur un modèle similaire à *Onion routing* [1] et utilise une architecture pair-à-pair. *RAC* force les nœuds rationnels à suivre le protocole en permettant à chaque nœud de surveiller anonymement le comportement des autres nœuds. Cette surveillance anonyme se fait grâce à un sous-protocole permettant la diffusion des messages de façon fiable en présence de nœuds rationnels. Pour finir, les nœuds sont séparés en groupe pour réduire le coût de chaque diffusion.

Nous avons simulé *RAC* à l'aide de Omnet++ [13] et comparé ses performances aux seuls protocoles de l'état de l'art capable de fonctionner en présence de nœuds rationnels : Dissent v1 [4] et Dissent v2 [5]. Notre évaluation montre que *RAC* assure un bien meilleur débit et une montée en charge parfaite là où le débit des deux versions de Dissent décroit avec le nombre de nœuds dans le système. Nous avons aussi évalué mathématiquement l'anonymat garanti par *RAC*, *Onion routing* [1] ainsi que les deux versions de Dissent. Cette évaluation montre que si l'anonymat garanti par *RAC* est inférieur à celui garanti par Dissent, il est aussi supérieur à celui garanti par *Onion routing*. Hors, ce dernier est le mécanisme d'anonymat le plus utilisé aujourd'hui, notamment au sein du système Tor [2].

2.3 Plan

La suite de cette partie est organisée ainsi. Le chapitre 3 présente les protocoles existants en matière de communications anonymes et explique pourquoi un nouveau protocole est nécessaire. Nous décrivons ensuite le protocole *RAC* en chapitre 4, avant de prouver son anonymat et sa résistance aux nœuds rationnels dans le chapitre 5. Pour finir, nous évaluons ses performances au chapitre 6.

3

État de l'art

Sommaire

De nombreux protocoles de communications anonymes ont été proposés au cours de ces dernières années. Ces protocoles peuvent être divisés en deux principales catégories : ceux basés sur DC-Net [3] et ceux basés sur Mix [22]. Il existe quelques protocoles ne se basant pas sur ces deux protocoles fondateurs, mais ils supposent un adversaire moins fort que celui choisi pour cette étude. Nous diviserons donc cette section en trois parties, une première partie, section 3.1 présentant DC-Net et les principaux protocoles en découlant, une deuxième section, section 3.2, présentant Mix et les principaux protocoles en découlant et pour finir une section présentant un protocole ne s'appuyant ni sur les idées de DC-Net, ni sur celles de Mix, section 3.3.

3.1 DC-Net et ses descendants

3.1.1 DC-Net

L'un des chercheurs ayant posé les fondements des protocoles de communications anonymes est David Chaum. Il a notamment proposé le protocole Dining Cryptographers networks (DC-Nets) [3]. Ce protocole a l'avantage d'assurer un anonymat mathématiquement incassable, même lorsque l'adversaire a une capacité de calcul infinie.

Afin d'illustrer son protocole David Chaum s'est appuyé sur le problème suivant. Un groupe d'experts en cryptographie est assis autour d'une table au restaurant. A la fin de leur repas, le serveur les informe que leur note a déjà été réglée et que la personne ayant réglé la note souhaite garder l'anonymat. Les experts pensent que leur note a, soit été payée par l'un d'entre eux, soit été payée par la National Security Agency (NSA). Les experts aimeraient savoir si la NSA a payé leur note, mais ne veulent pas briser l'anonymat de celui d'entre eux qui pourrait avoir payé le repas. Si l'un des experts a payé la note, peut-il le faire savoir aux autres sans briser son anonymat ?

La réponse est oui. Pour ce faire les experts doivent procéder comme illustré sur la figure 3.1. En étape 1 chaque expert effectue un tirage à pile ou face. Ce tirage est fait secrètement et chaque expert mémorise la valeur 0 si le résultat est face et 1 si le résultat est pile. Cette valeur est appelée un secret. En étape 2 chaque expert transmet, de façon privée, son secret à l'expert assis à sa gauche. Connaissant sa propre valeur x et celle y de son voisin de gauche chaque expert calcule $z = x \oplus y$, avec $\oplus$ la somme modulo 2 (XOR). Si l'un des experts, A dans l'exemple, a payé la note, il calcule $z = x \oplus y \oplus 1$ au lieu de $x \oplus y$. Le résultat de ces calculs est illustré par la valeur qu'ont les nœuds en étape 3. Par exemple E n'a pas payé la note, il calcule donc : $z = x \oplus y$ avec $x = 1$ et $y = 0$ ce qui donne $z = 1 \oplus 0 = 1$. A a payé la note, il calcule donc $z = x \oplus y \oplus 1$ avec $x = 1$ et $y = 0$, il obtient donc $z = 1 \oplus 0 \oplus 1 = 0$ Une fois ce calcul effectué chaque expert annonce la valeur de z qu'il a obtenu à l'ensemble des experts (étape 3). Pour finir les experts calculent $R = z_1 \oplus z_2 \oplus \ldots \oplus z_n$ avec $z_1, z_2, \ldots z_n$ les valeurs annoncées par chacun des experts. Ainsi, en figure 3.1, les nœuds calculent $R = 0 \oplus 0 \oplus 1 \oplus 0 \oplus 1 \oplus 1 = 1$.

Si $R = 1$ cela signifie que l'un des experts a payé la note. Si $R = 0$ cela signifie que la NSA a payé la note. Le raisonnement aboutissant à ce résultat est le suivant : si l'information secrète indiquant si l'expert i a payé la note est notée s_i et le résultat du pile ou face de i est noté r_i, chaque expert calcule $z_i = r_i \oplus r_{i-1} \oplus s_i$. Ce qui donne :

$$R = z_1 \oplus z_2 \oplus \ldots \oplus z_n \tag{3.1}$$

$$= (r_1 \oplus r_n \oplus s_1) \oplus (r_2 \oplus r_1 \oplus s_2) \oplus \ldots \oplus (r_n \oplus r_n - 1 \oplus s_n) \tag{3.2}$$

$$= r_1 \oplus r_1 \oplus s_1 \oplus r_2 \oplus r_2 \oplus s_2 \oplus \ldots \oplus r_n \oplus r_n \oplus s_n \tag{3.3}$$

$$= s_1 \oplus s_2 \oplus \ldots \oplus s_n \tag{3.4}$$

Le passage de 3.2 à 3.3, se fait par simple réorganisation des termes. Cette réorganisation est possible car $\oplus$ est commutatif. Le passage de 3.3 à 3.4 découle du fait que $\forall x, x \oplus x = 0$. Si aucun expert n'a payé la note, tous les s_i sont égaux à 0, R vaut donc 0. Si un expert a payé la note, tous les s_i sont égaux à 0 sauf celui de l'expert qui a payé, R est donc égal à 1.

Ce problème illustre le principe de base de DC-Net. Il montre comment un nœud peut communiquer un bit d'information à l'ensemble des autres nœuds sans qu'aucun autre nœud ne sache qui a envoyé ce bit. Il est facile d'imaginer comment étendre ce mécanisme à l'envoi de messages de plusieurs bits. Cependant, ce problème n'est qu'une illustration qui ne suffit pas. Il est en effet facile de briser l'anonymat garanti par le protocole tel que présenté ci-dessus. En effet, si l'addition a été payée par l'expert j et si les experts $j - 1$ et $j + 1$ sont malveillants, ils peuvent communiquer entre eux pour connaitre r_j, r_{j-1} et $r_j + 1$. Ils peuvent alors calculer la valeur z_j que j devrait

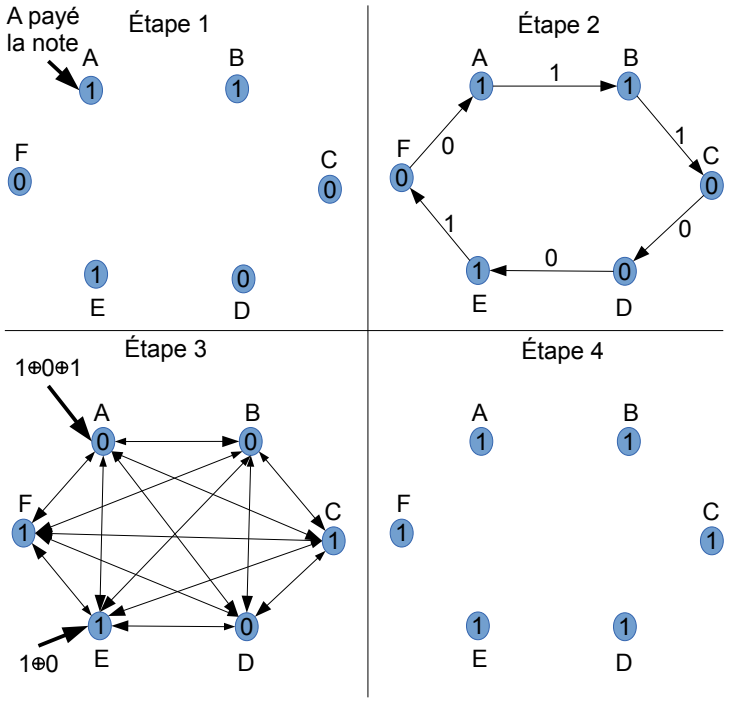


FIGURE 3.1 – Fonctionnement de DC-Net

annoncer et déduire que j a payé en s'appuyant sur la différence entre la valeur z_j qu'ils ont calculé et la valeur annoncée par j. Pour éviter ce problème il est possible d'augmenter le nombre de nœuds avec lesquels chaque nœud partage un secret : chaque nœud sélectionne n nœuds (par exemple ses n voisins de gauche), effectue un tirage de pile ou face par nœud et indique le résultat de ce tirage au nœud correspondant. De cette façon, il faut que l'ensemble des nœuds ayant partagé un secret avec j et l'ensemble des nœuds avec lesquels j a partagé un secret collaborent entre eux pour connaitre la valeur que j devrait annoncer.

DC-Net permet donc d'assurer qu'il est impossible de briser l'anonymat d'un nœud sans contrôler tous les autres nœuds du système. En effet, si lors de chaque envoi de messages, chaque nœud du système partage un secret avec chaque autre nœud du système, tous les nœuds du système devront collaborer pour déterminer ce qu'envoie un nœud. Cependant, DC-Net a deux inconvénients majeurs. Tout d'abord son coût en bande passante : pour qu'un nœud transmette anonymement un bit d'information il faut que chaque nœud envoie un bit secret à chaque autre nœud, il faut donc que $n(n-1)$ bits circulent sur le réseau. Il faut ensuite que chaque nœud transmette la valeur qu'il a calculé pour z à chaque autre nœud, $n(n-1)$ bits circulent donc de nouveau sur le réseau. Au total, l'envoi anonyme d'un bit, coûte $2n(n-1)$ bits de bande passante. Ce surcoût réseau rend le protocole inutilisable lorsque le nombre de nœuds grandit. Le second inconvénient de DC-Net est qu'il souffre d'un problème de collisions de messages : si deux nœuds essayent de transmettre un bit d'information au même moment, la valeur de R sera égale à la somme modulo 2 des bits d'information transmis par chacun des deux nœuds. Il sera alors impossible de récupérer les messages.

Ce problème de collision permet d'effectuer une attaque par déni de service en ne contrôlant qu'un nœud dans le système. Certain travaux ont été effectués pour éviter les collisions et détecter les attaques par dénis de service [23, 24], mais le coût en bande passante de ces protocoles est prohibitif.

3.1.2 Herbivore

Le protocole Herbivore [6] a pour but de rendre DC-Net utilisable en pratique en supprimant les limitations. Pour ce faire DC-Net divise l'ensemble des nœuds en groupes de taille au moins k et au plus $3k$. Lorsque les nœuds veulent communiquer ils utilisent DC-Net au sein du groupe auquel ils appartiennent, et si le nœud auquel le message est destiné n'appartient pas à ce groupe, un nœud est désigné pour faire suivre le message d'un groupe à l'autre.

Afin d'éviter les collisions, Herbivore propose aussi un mécanisme de réservation anonyme de créneaux d'envoi. Herbivore fonctionne par tours. Au début de chaque tour un processus permet aux nœuds de réserver des créneaux d'envoi. Chaque nœud utilise ensuite le créneau qu'il a réservé pour envoyer un message de taille prédéfinie par le système. Pour réserver un créneau chaque nœud choisi un chiffre c entre 1 et m, m étant grand par rapport au nombre de nœuds dans le système. Chaque nœud génère ensuite un vecteur de taille m dont toutes les valeurs sont à 0 sauf la valeur d'indice c qui est à 1. Si un nœud ne veut pas transmettre d'information durant un tour, il crée un vecteur ne contenant que des 0. Les nœuds transmettent ensuite simultanément leur vecteur à l'ensemble des autres nœuds. Afin qu'il soit impossible de lier un nœud à son vecteur, cette transmission se fait en utilisant DC-Net. Le résultat de cette transmission est donc un XOR de tous les vecteurs. Chaque nœud sait alors que si la valeur d'indice c du vecteur final est 1 il y a une forte probabilité pour qu'il soit le seul à avoir réservé ce créneau, il peut donc transmettre son message lorsque le tour de ce créneau arrive.

Ces deux mécanismes permettent de diminuer le coût réseau du protocole et de diminuer la probabilité de collisions accidentelles de messages. Cependant, Herbivore ne règle pas le problème des collisions volontaires que pourrait créer un nœud malveillant : Herbivore se contente de proposer aux nœuds du groupe attaqué de quitter le système et de rejoindre un autre groupe. De plus, Herbivore ne prend pas en compte les nœuds rationnels. Ces derniers peuvent par exemple tenter de réserver tous les créneaux d'envoi. De cette façon ils augmentent leur chance d'obtenir un créneau libre et peuvent tenter d'envoyer plusieurs messages par tour. Un tel comportement aurait pour conséquence d'empêcher tout nœud de transmettre des données et rendrait le protocole inutilisable.

3.1.3 Dissent

Le protocole DC-Net étant peu efficace et souffrant de problèmes de collisions, il a longtemps été ignoré au profit des protocoles basés sur Mix que nous verrons en section 3.2. Cependant, il est revenu sur le devant de la scène en 2010, lorsque, constatant qu'il n'existait aucun protocole de communications anonymes fonctionnant en présence de nœuds rationnels, B. Ford & al. ont proposé un protocole se basant sur DC-Net et obligeant les nœuds à rendre compte de leurs actes. Dans, ce protocole, appelé Dissent [4], les nœuds peuvent être réprimandés s'ils ne suivent pas correctement le protocole. Une première version, que nous appellerons Dissent v1 [4], a été publiée

en 2010. Celle-ci ne fonctionne pas avec plus de 40 nœuds. Une deuxième version, que nous appellerons Dissent v2 [5], a été développée afin de pouvoir atteindre un nombre de nœuds plus conséquent. Cette deuxième version a été publiée en 2012.

3.1.3.1 Dissent v1

Le protocole Dissent v1 [4] se compose de deux parties : une première partie se basant sur un mécanisme de mélange vérifiable et une deuxième se basant sur DC-Net. La première partie pourrait suffir pour communiquer anonymement, mais comme nous allons le voir, cela serait peu efficace. Elle est donc utilisée pour partager les secrets utilisés dans la partie DC-Net et pour réserver des créneaux d'envois de messages nécessaires à la deuxième partie.

Le mécanisme de mélange vérifiable fonctionne comme suit. Tous les nœuds du système sont ordonnés et ont deux jeux de clés de chiffrement appelés jeu primaire et jeu secondaire. À chaque ronde chaque nœud chiffre le message qu'il veut envoyer, d'abord en y appliquant chacune des clés secondaires publiques des nœuds du système (dans l'ordre inverse de celui des nœuds), puis en y appliquant chacune des clés primaires publiques des nœuds du système (dans l'ordre inverse de celui des nœuds). Chaque nœud envoie ensuite le message chiffré au premier nœud du système. Une fois qu'il a reçu tous les messages, le premier nœud déchiffre la première couche de chiffrement à l'aide de sa clé primaire privée, mélange les messages et les envoie au second nœud. Le second nœud déchiffre alors la deuxième couche de chiffrement, mélange les messages et les envoie au troisième nœud. Les messages circulent ainsi jusqu'à ce qu'ils atteignent le dernier nœud et que toutes les couches de chiffrement primaire aient été déchiffrées. Le dernier nœud envoie alors l'ensemble des messages à l'ensemble des nœuds du système. A la réception des messages, chaque nœud vérifie que son message est bien présent et qu'il n'a pas été altéré. Si c'est le cas, le nœud envoie à tous les autres nœuds un message indiquant qu'il est d'accord pour continuer le processus. Si ce n'est pas le cas, il l'indique aux autres nœuds. Si tous les nœuds sont d'accord pour continuer le processus, chaque nœud révèle sa clé privée secondaire à l'ensemble des autres nœuds. Les nœuds peuvent alors déchiffrer les couches de chiffrement secondaire et obtenir les messages. Si un nœud indique ne pas vouloir continuer le processus, chaque nœud détruit son jeu de clés secondaire et révèle son jeu de clés primaires. De cette façon chaque nœud peut rejouer la séquence des déchiffrements et mélanges et ainsi trouver le nœud qui a altéré un message.

Ce mécanisme pourrait suffire à envoyer des messages de façon anonyme, mais il faudrait pour cela que tous les messages aient la même taille. En effet, si les messages ont des tailles différentes, il est possible d'identifier chaque message par sa taille et de savoir la position de chaque message après chaque mélange. Il est possible d'obliger les nœuds à envoyer des messages de taille standardisée. Cependant, imposer des messages de même taille dans un mécanisme par ronde peut s'avérer coûteux en termes de latence (un nœud voulant envoyer un gros message doit le faire en plusieurs rondes) et de bande passante (un nœud voulant envoyer de petits messages use beaucoup de bande passante en enrobage du message). Afin d'éviter ce problème Dissent v1 propose d'utiliser un mécanisme à la DC-Net. Le mécanisme précédent est utilisé pour faire parvenir de façon anonyme des "graines" à l'ensemble des nœuds du système : chaque message contient une graine par nœud ainsi que la taille du message à envoyer par DC-Net.

Chaque graine est chiffrée de façon à ne pouvoir être lue que par le nœud auquel elle est destinée. Les messages envoyés à l'aide du mécanisme de mélange vérifiable sont traités dans l'ordre dans lequel ils sont envoyés par le dernier nœud. Pour chaque message, chaque nœud déchiffre la graine qui lui correspond et génère une chaîne de caractères aléatoires à l'aide de cette graine, d'un générateur de nombres pseudo aléatoires et de la taille du message. Cette chaîne de caractères correspond au secret de DC-Net. Le nœud envoyant le message génère l'ensemble des secrets, les somme à l'aide d'une somme modulo 2 et y ajoute son message à l'aide de la même somme. Chaque nœud envoie ensuite la chaîne qu'il a obtenue à l'ensemble des nœuds. Pour finir, une fois toutes les chaînes reçues, les nœuds peuvent les sommer modulo 2 pour obtenir le message. Si un nœud n'envoie pas la bonne chaîne, le message ne sera pas déchiffrable. Une nouvelle ronde du mécanisme de mélange vérifiable sera alors lancée pour permettre à l'émetteur du message de dénoncer le nœud n'ayant pas envoyé la bonne chaîne.

L'ensemble des deux mécanismes précédents permet de s'assurer qu'aucun nœud ne pourra dévier du protocole et donc qu'aucun nœud ne pourra en impacter les performances en étant rationnel. En contrepartie, pour qu'un nœud envoie un bit il faut, pendant la phase DC-Net, que l'ensemble des nœuds du système envoient un message de 1 bit à l'ensemble des autres nœuds du système. Ceci est équivalent à faire N diffusions dans un système contenant N nœuds. Nous noterons le coût de ces diffusions $N \times Bcast(N)$, avec $Bcast(x)$ le coût d'une diffusion dans un système contenant x nœuds. Afin de faciliter la comparaison, nous limiterons le coût de Dissent v1 à $N \times Bcast(N)$, mais il ne faut pas oublier que ce coût s'ajoute au coût du mélange vérifiable qui nécessite le passage de N messages (un par nœud) à travers l'ensemble des nœuds du système. Ces messages sont certes petits (ils contiennent seulement des graines) mais la latence du passage des messages à travers l'ensemble des nœuds n'est pas négligeable. Le coût en bande passante et en temps des deux mécanismes constituant Dissent v1 est tel que, comme indiqué dans l'article présentant Dissent v1 [4], le protocole n'est pas utilisable avec plus de 40 nœuds. Afin d'illustrer ce problème nous avons effectué une simulation à l'aide d'Omnet++ [13] dans laquelle tous les nœuds sont liés entre eux par des liens de 1Gb/s. Nous avons mesuré le débit de Dissent v1 en fonction du nombre de nœuds. Le résultat est présenté en figure 3.2. Nous observons que le débit chute très rapidement lorsque le nombre de nœuds augmente et qu'il devient inutilisable dès 40 nœuds.

3.1.3.2 Dissent v2

Afin de pouvoir supporter un nombre de nœuds plus significatif Wolinsky et al. ont proposé une nouvelle version de Dissent, Dissent v2 [5]. Cette version réduit le nombre de messages échangés en s'appuyant sur un modèle client-serveur : les nœuds qui veulent envoyer des messages sont les clients ; les nœuds assurant l'anonymat sont les serveurs. Le protocole fonctionne en deux phases : une phase de mélange vérifiable et une phase d'envoi anonyme de messages. Lors de la phase de mélange vérifiable chaque client soumet une graine par serveur. À la fin de ce processus chaque serveur connait les graines qui lui correspondent et l'ordre dans lequel elles vont être utilisées, mais aucun serveur ne peut faire un lien entre une graine et un client. Cette phase est suivie de plusieurs phases d'envoi de messages se basant sur le mécanisme de DC-Net. Lors de chaque phase, chaque client génère une chaîne pseudo-aléatoire à l'aide de chacune des

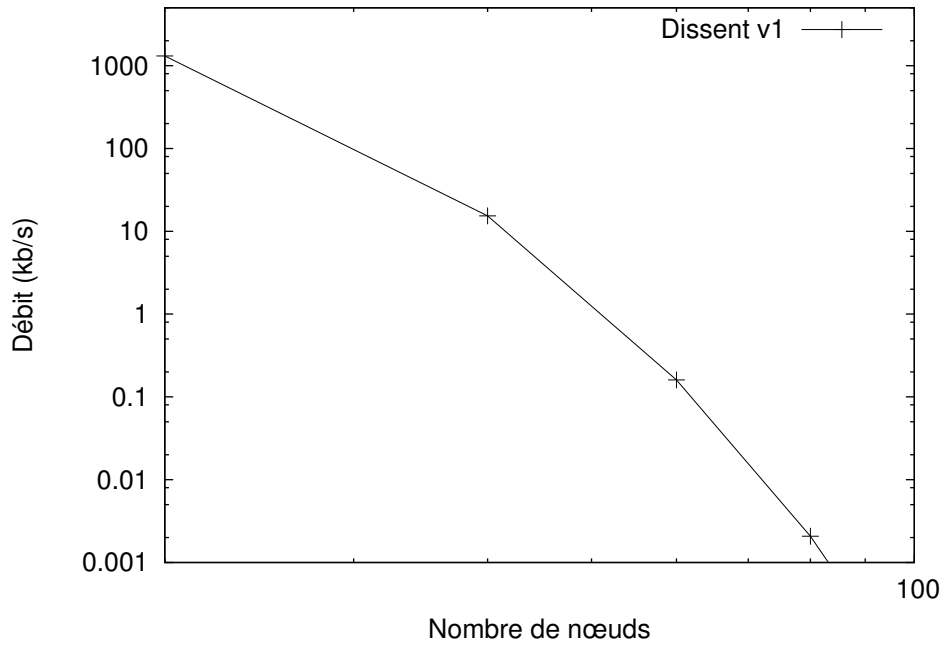


FIGURE 3.2 – Débit de Dissent v1 en fonction du nombre de nœuds

graines qu'il a fait parvenir aux serveurs pendant la phase de mélange vérifiable. Chaque client fait ensuite la somme modulo 2 des chaînes obtenues et y ajoute son message lorsque c'est son tour d'en envoyer un. Pour finir, les clients envoient la chaîne obtenue à un serveur. Les serveurs font une somme modulo 2 de l'ensemble des messages qu'ils ont reçu ainsi que des chaînes de caractères qu'ils ont générées à l'aide des graines qui leur ont été attribuées pendant la phase de mélange vérifiable. Ils envoient ensuite la chaîne obtenue à l'ensemble des autres serveurs. Pour finir, les serveurs somme modulo 2 les chaînes qu'ils ont reçues de l'ensemble des autres serveurs, obtiennent le message en clair et le font suivre à l'ensemble de leurs clients.

L'utilisation de serveurs permet de réduire le nombre de messages échangés : les messages ne doivent passer que par l'ensemble des serveurs et non par l'ensemble des nœuds du système. Cette réduction du nombre de messages échangés permet à Dissent v2 de supporter un nombre bien plus important de nœuds que Dissent v1. La figure 3.3 montre ainsi que Dissent v2 permet de monter jusqu'à 1000 nœuds. Cependant, la figure montre aussi que la montée en charge de Dissent v2 reste loin d'être optimale. Cela vient de la charge en bande passante que doivent supporter les serveurs : pour qu'un client envoie un bit de donnée, chaque client doit envoyer un bit de donnée. Chaque serveur doit donc recevoir un bit de donnée par client à sa charge. S'il y a N clients, M serveurs et que les clients sont équitablement répartis sur les serveurs, chaque serveur devra recevoir $\frac{N}{M}$ bits pour qu'un client envoie un bit. Chaque serveur devra aussi transmettre le message à l'ensemble de ses clients à la fin de la ronde. Chaque serveur devra donc envoyer $\frac{N}{M}$ bits à ses clients. Ces opérations correspondent à des diffusions dans un groupe contenant $\frac{N}{M}$ nœuds, elles ont donc un coût cumulé de $2 \times Bcast(\frac{N}{M})$. Ces nombreux échanges peuvent saturer le réseau entre les clients et les

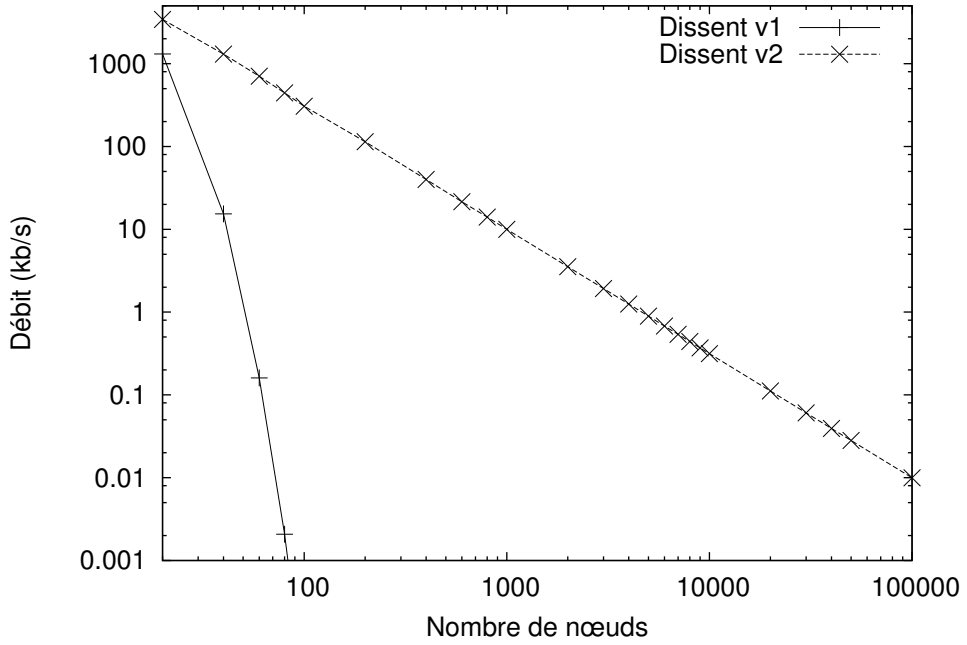


FIGURE 3.3 – Débit de Dissent en fonction du nombre de nœuds

serveurs. Afin de réduire la saturation, il est possible de réduire le coût de $Bcast(\frac{N}{M})$ en augmentant le nombre de serveurs. De cette façon le nombre de nœuds concernés par chaque diffusion est réduit. Cependant, pour qu'un client envoie un bit, chaque serveur doit faire parvenir un bit à l'ensemble des autres serveurs. Ceci revient à M diffusions dans un groupe de taille M, soit un coût de $M \times Bcast(M)$. Ces nombreuses diffusions aboutissent à la situation suivante : plus le nombre de serveurs est grand plus le lien liant un serveur aux autres serveurs est saturé. Pour résumer, le coût total de Dissent v2 est d'au moins $2 \times Bcast(\frac{N}{M}) + M \times Bcast(M)$. La première moitié de ce coût augmente avec N et diminue avec M, la seconde moitié augmente avec M. Il est donc possible, pour chaque valeur de N, de trouver une valeur de M minimisant le coût total. Cependant, la fonction de coût est croissante avec N, ce qui induit une faible possibilité de monter en charge du protocole. Ceci est illustré sur la figure 3.3. Dans cette figure, l'axe des abscisses représente le nombre de nœuds voulant communiquer anonymement, c'est à dire le nombre de clients, N. L'axe de ordonnés représente le débit auquel les nœuds peuvent envoyer des messages. Afin d'obtenir cette figure, nous avons choisit, Pour chaque valeur de N, le nombre de serveurs M maximisant le débit du système, c'est à dire la valeur de M minimisant le coût total de Dissent v2 : $2 \times Bcast(\frac{N}{M}) + M \times Bcast(M)$.

Il faut aussi souligner que l'utilisation de serveurs est un point faible. En effet, même si Dissent v2 est construit de façon à ce que l'existence d'un unique serveur bienveillant assure l'anonymat des clients, les évolutions récentes en matière de sur-

veillance informatique [1] [2] [3] peuvent nous faire douter de l'existence et de la viabilité d'un tel serveur. Ceci d'autant plus que le nombre de serveurs doit rester faible pour que les performances restent bonnes.

3.2 Mix et ses descendants

3.2.1 Mix

En plus de DC-Net, David Chaume a proposé un autre article fondateur dans le domaine des communications anonymes : Chaume Mix [22]. Dans cet article, Chaume introduit l'idée d'un nœud central dit "mix", qui dissimule la correspondance entre les messages qu'il reçoit et les messages qu'il envoie, et par lequel vont passer tous les messages. Cette idée sera ensuite reprise dans de nombreuses implémentations commerciales telles que anon.penet.fi [25], Cypherpunk remailers [26] et Mixminion [27].

Le mix possède un jeu de clés de chiffrement asymétrique dont la clé publique est connue de tous les nœuds. Lorsqu'un nœud A veut envoyer un message à un nœud B, il chiffre le message et l'adresse de B à l'aide de la clé publique du mix, puis envoie le tout au mix. Un salage est effectué, pour éviter que l'adversaire ne puisse détecter les messages identiques, et les messages sont enrobés, pour qu'il soit impossible à l'adversaire de se baser sur la taille des messages pour les reconnaitre. Si A connait la clé publique de B, A peut commencer par chiffrer son message avec la clé de B afin d'empêcher le mix de connaitre le contenu du message. Lorsque le mix reçoit un message il le déchiffre et le met dans un groupe en attendant d'avoir reçu un nombre suffisant de messages (nous verrons ci-dessous ce qui est entendu par "assez"). Une fois qu'il a reçu assez de messages le mix fait suivre les messages à leurs destinataires.

Les messages étant déchiffrés par le mix, les messages entrant et sortant du mix n'ont pas le même contenu. La cryptographie garantit que l'adversaire ne peut pas effectuer l'opération de déchiffrement sur les messages reçus par le mix. L'adversaire ne peut donc pas faire de corrélation entre les messages entrant et sortant du mix, en se basant sur leur contenu. En principe, le fait de n'envoyer les messages qu'une fois en avoir reçu assez permet d'éviter que l'adversaire ne puisse faire une corrélation temporelle entre les messages entrant dans le mix et les messages en sortant. Mais nous verrons ci-dessous que c'est rarement le cas.

Le premier défaut de cette solution est qu'elle est centralisée. Il faut donc que les nœuds aient une forte confiance vis-à-vis du mix ; si le mix est compromis le système ne fournit plus d'anonymat. Une solution pour réduire ce problème est d'utiliser plusieurs mixs les uns à la suite des autres [28, 29, 30, 31] : A envoie son message à un premier mix, qui le fait suivre à un second, qui le fait suivre à un troisième, etc. Jusqu'à un dernier mix qui délivre le message au destinataire. De cette façon l'adversaire doit compromettre tous les mixs pour casser l'anonymat du système. Le choix de l'ordre des mixs peut être fixe [30] (l'ordre est le même pour tous les messages et tous les nœuds) ou libre [31] (l'ordre est choisi par chaque client au moment d'envoyer son message).

Le second défaut de cette solution vient de la façon dont le ou les mixs décident qu'ils ont reçu assez de messages pour les faire suivre à leurs destinataires. En effet, si

1. Grand firewall de chine : http ://fr.wikipedia.org/wiki/Grand_Firewall_de_Chine
2. Usa patriot act : http ://fr.wikipedia.org/wiki/Patriot_Act
3. Prism : http ://fr.wikipedia.org/wiki/PRISM_%28programme_de_surveillance%29

l'adversaire parvient à connaitre et compromettre ce mécanisme il peut être capable de faire des corrélations temporaires entre les messages entrant dans le mix et les messages en sortant. Les premiers mécanismes se basaient sur le nombre de messages reçus [22] ou sur un intervalle de temps régulier [32] pour déterminer si assez de messages avaient été reçus. Des attaques dites "attaques $(n-1)$" et "attaques par égouttement" ont vite fait leur apparition pour mettre à mal ces mécanismes et de nombreux autres mécanismes ont été proposés [32, 33, 34, 35, 36]. Cependant, tous les mécanismes existants sont vulnérables à différentes attaques actives [32, 37].

Pour finir, cette solution soufre d'un problème de montée en charge et ne fournit que l'anonymat des relations. Le problème de montée en charge est dû au fait que tous les messages doivent passer par le ou les mixs. Les mixs forment donc un goulot d'étranglement. La limitation d'anonymat vient du fait qu'un adversaire capable d'écouter tout le réseau est capable à tout moment de savoir qui envoie quel message et qui reçoit quel message, il n'y a donc pas d'anonymat des émetteurs et des destinataires. Il y a cependant un anonymat des relations, car l'adversaire ne peut faire le lien entre les messages envoyés et les messages reçus.

3.2.2 Onion routing

3.2.2.1 Premier protocole

Le protocole Mix présenté ci-dessus ne peut être utilisé pour des applications en temps réel telles que surfer sur Internet ou le SSH. En effet, le nœud mix attend d'avoir reçu assez de messages avant de les faire suivre. Cette attente, ajoutée aux temps de chiffrements et de déchiffrements, entraîne d'importantes latences. C'est pour réduire ces latences pour des applications temps réel que le NRL (U.S Naval Research Laboratory) a développé *Onion routing* [1].

A l'instar d'une solution mix multiserveurs, *Onion routing* s'appuie sur l'existence de serveurs qui relaient les messages. La différence avec mix réside en deux points :

— Les serveurs ne sont pas des mixs ; ils relaient les messages aussitôt qu'ils les ont déchiffrés. L'anonymat s'appuie sur le nombre de serveurs, le nombre de routes par lesquels peut passer un message et la densité du trafic.

— Dans *Onion routing*, la majorité des chiffrements sont faits à l'aide de clés symétriques alors que tous les chiffrements de Mix sont faits à l'aide de clés asymétriques.

Dans *Onion routing*, lorsqu'un nœud A veut envoyer un message à un nœud B, il commence par créer un circuit de communication partant de A, passant par plusieurs serveurs et arrivant à B. Il utilisera ensuite ce circuit pour l'ensemble des messages qu'il enverra à B. L'établissement d'un circuit, préalablement à la communication, permet d'utiliser des clés de chiffrement symétriques lors de l'envoi des messages de A vers B.

La création d'un circuit est illustrée sur la figure 3.4 : A sélectionne l serveurs qui constitueront le circuit (C, E et F dans la figure). Pour B et chacun des serveurs, A choisit deux clés de chiffrement symétriques : une clé vers l'avant et une clé vers l'arrière. Les couples de clés sont noté C_i dans la figure. Les clés vers l'avant seront utilisées pour chiffrer les messages allant de A vers B et les clés vers l'arrière serviront à chiffrer les réponses de B vers A. Une fois les serveurs et les clés choisis, A chiffre les clés destinées à B (C_B) à l'aide de la clé publique de B et obtient $\{C_B\}_B$. A chiffre ensuite $\{C_B\}_B$,

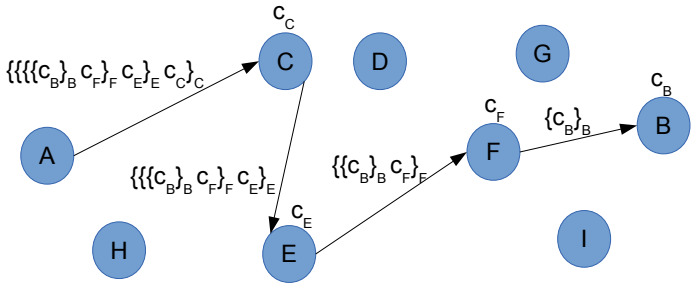


FIGURE 3.4 – Fonctionnement d'*Onion routing*

l'identité de B et les clés destinées au dernier serveur du circuit (C_F) à l'aide de la clé publique de celui-ci. Il obtient alors$\{\{C_B\}_B C_F\}_F$. Il chiffre ensuite $\{\{C_B\}_B C_F\}_F$, l'identité du dernier serveur ainsi que les clés destinées à l'avant dernier serveur du circuit avec la clé publique de celui-ci et réitère ainsi l'opération pour tous les serveurs du circuit. Une fois l'opération terminée A envoie $\{\{\{\{C_B\}_B C_F\}_F C_E\}_E C_C\}_C$ au premier serveur. Le premier serveur déchiffre $\{\{\{\{C_B\}_B C_F\}_F C_E\}_E C_C\}_C$ à l'aide de sa clé privée. Il obtient ainsi ses clés symétriques, C_C, l'identité du serveur qui le suit et $\{\{\{C_B\}_B C_F\}_F C_E\}_E$. Il envoie ensuite $\{\{\{C_B\}_B C_F\}_F C_E\}_E$ au serveur qui le suit. Le message circulera ensuite de serveur en serveur jusqu'à atteindre B. Lorsque le message atteint B, tous les serveurs connaissent les clés symétriques utilisées dans le circuit ainsi que les serveurs se trouvant directement avant et après eux dans le circuit. Le circuit est alors établi.

Pour envoyer un message, A chiffre le message avec la clé symétrique qu'il a fait parvenir à B. Il chiffre ensuite ce message chiffré avec la clé vers l'avant du dernier serveur et réitère ainsi pour l'ensemble des serveurs jusqu'au premier serveur. Le message est alors protégé par des couches de chiffrements s'apparentant aux couches d'un oignon, d'où le nom du protocole. A envoie ensuite le message au premier serveur qui le déchiffre, le enrobe et le fait suivre au serveur suivant. L'opération se répète ainsi jusqu'à ce que le message atteigne B. Lorsque B veut répondre à A il chiffre son message avec sa clé vers l'arrière et l'envoie au dernier serveur. Celui-ci chiffre le message avec sa clé vers l'arrière et l'envoie à l'avant-dernier serveur, et ainsi de suite.

Le fait de construire des oignons assure que chaque serveur ne connait que l'identité du serveur qui le précède et du serveur qui le suit dans le circuit. En effet, lorsqu'un serveur reçoit un oignon il est en mesure de savoir quel serveur lui a envoyé le message, mais, si le serveur précédent n'est pas malicieux et ne lui transmet pas d'information, il n'est pas en mesure de savoir quel autre serveur se trouve avant dans le circuit. De la même manière, le serveur sait qui est le serveur suivant dans le circuit, mais il est incapable de déchiffrer la couche suivante de l'oignon ; il ne peut donc pas connaître le reste du circuit. Il suffit donc qu'un circuit contienne un serveur non contrôlé par l'adversaire pour que l'adversaire ne puisse connaitre les deux extrémités du circuit. En effet, même s'il contrôle tous les nœuds du circuit avant et après le nœud bienveillant, l'adversaire ne pourra faire le lien entre les messages que ses nœuds envoient au nœud bienveillant et les messages que ses nœuds reçoivent du nœud bienveillant : les messages que ses nœuds envoient au nœud bienveillant peuvent être dirigés vers une

autre fin de circuit et réciproquement les messages que ses nœuds reçoivent peuvent venir d'un autre début de circuit. Ceci permet à *Onion routing* d'assurer l'anonymat des relations. Si le nœud émetteur fait aussi office de serveur, *Onion routing* assure aussi l'anonymat des émetteurs, car il n'est pas possible de savoir si l'émetteur est en train d'envoyer un message ou d'en relayer un. *Onion routing* peut aussi assurer l'anonymat des destinataires. Pour ce faire le nœud A, qui veut voir son anonymat assuré, crée un circuit et s'arrange pour que les autres nœuds connaissent la fin de ce circuit. Lorsqu'un nœud B veut communiquer avec A il se connecte à la fin du circuit pour envoyer ses messages.

Si le nombre de serveurs est assez élevé, le nombre de circuits possibles devient tel qu'il est impossible pour l'adversaire d'analyser le trafic. Cependant, depuis la création d'*Onion routing* il a été suspecté puis prouvé qu'en l'absence d'un fort trafic, un opposant passif a la faculté d'identifier des motifs dans le trafic lui permettant de suivre le parcours d'un message. Ces attaques ont été appelées "attaques temporelles" et "attaques bout-en-bout" et ont été grandement étudiées [38, 39, 40, 41, 42, 43, 44]. Pour contrer ces attaques, la NRL créa une nouvelle version de *Onion routing* en y ajoutant des méthodes de management du trafic telles que le enrobage des messages ou la régulation du taux d'émission, ainsi que du mix temps réel [45, 46]. Mais ces solutions ne s'avérèrent pas efficaces et furent supprimées dans la version suivante d'*Onion routing*.

3.2.2.2 Tor

En 2003, le NRL a mis sur pied une nouvelle version d'*Onion routing* appelée Tor [2]. Tor permet de transmettre tout type de communication TCP et est optimisé pour le trafic Internet.

Pour communiquer, un client Tor choisit trois serveurs au hasard afin de créer un circuit de communication par lequel passeront ses messages. Dans cette nouvelle version, le client ne construit pas un oignon lorsqu'il veut créer un circuit. Il se connecte au premier serveur puis demande à ce serveur de se connecter au serveur suivant et ainsi de suite. Chacune de ces connexions est faite à l'aide d'un échange de clés Diffie-Hellman, ce qui garantit l'anonymat des communications tout en garantissant une résistance aux attaques par contrainte (l'opposant contraint un serveur à lui donner ses clés), car seules des clés à faible durée de vie sont utilisées. Notez que l'authentification des routeurs se fait toujours grâce à des clés asymétriques.

Comme *Onion routing*, Tor n'assure pas d'anonymat lorsque le débit de messages dans le réseau est faible. Le design de Tor cherche donc à garantir l'anonymat à travers sa grande utilisabilité et sa facilité d'utilisation [47, 48] qui lui garantissent une adoption large, entrainant un trafic important. Un fort travail a été fait sur l'interface utilisateur des clients, ainsi que sur la facilité à déployer un serveur. Le travail sur les serveurs permet à chacun de facilement déployer son propre serveur, ce qui garantit l'existence de nombreux serveurs contrôlés par différentes entités. Ceci permet de rendre plus difficile à un adversaire de contrôler une proportion importante de serveurs. Ceci facilite aussi la montée en charge du système. En effet, toutes les requêtes devant passer par trois serveurs, plus il y a de clients plus il faut de serveurs pour soutenir la charge. Ces choix de simplification furent judicieux puisque Tor est aujourd'hui le réseau de communications anonymes le plus connu et le plus utilisé [48, 49].

Des travaux ont de plus été effectués, en parallèle du développement de Tor, pour détecter les nœuds susceptibles de conduire une attaque [50, 51], mais ces approches demandent une latence élevée ou une quantité importante de faux trafic servant à couvrir le trafic réel. L'impact sur les performances étant contraire aux objectifs de Tor, ces travaux n'ont pas été retenus dans son développement.

3.2.2.3 Onion routing pair-à-pair

Dès le début des travaux sur Mix il a été admis que si tous les nœuds avaient à la fois un rôle de client et un rôle de serveur, cela renforcerait l'anonymat garanti par le système. Les principales difficultés adressées par les solutions développées en ce sens [52, 53, 54, 55, 56, 57, 58] sont la découverte du système et les connexions et déconnexions des nœuds. Le premier problème a notamment été adressé par Tarzan [52] qui propose de découvrir les nœuds présents dans le réseau grâce à un système basé sur un protocole de rumeurs. Le second problème, pour lequel de nombreux protocoles [57, 58, 59, 60] ont été proposés, est très important car à chaque fois qu'un nœud se déconnecte un serveur disparait. Tous les nœuds utilisant ce serveur doivent donc reconstruire un nouveau circuit de communication. Ceci pose deux problèmes : (1) le temps de détection de la rupture d'un circuit, ajouté au temps de construction d'un nouveau circuit, entraine une augmentation de la latence et (2) à chaque fois qu'un nœud construit un nouveau circuit de communication il augmente la probabilité de construire un circuit ne contenant que des nœuds adverses. L'adversaire peut donc choisir de déconnecter et reconnecter certains de ses nœuds dans le but de forcer les clients à créer de nouveaux circuits et à risquer de construire un circuit ne garantissant pas leur anonymat. L'une des solutions les plus utilisées pour éviter la rupture du circuit et de réunir les nœuds en petits groupes et de faire relayer les messages par un groupe de serveurs plutôt que par un serveur [57, 58, 60]. De cette façon lorsqu'un nœud quitte le système il est toujours possible d'utiliser les autres nœuds du groupe pour relayer les messages. Une autre solution proposée dans Tarzan [52] est de reconstruire le circuit à partir de l'endroit où un nœud est parti. De cette façon, seul un petit nombre de nœuds doit être changé, ce qui réduit la probabilité de construire un circuit ne contenant que des nœuds adversaires.

Une limitation des systèmes pair-à-pair non adressée par les protocoles existants est l'existence de nœuds rationnels. Chacun de ces protocoles compte en effet sur chaque nœud pour router les messages, hors il est possible que des nœuds décident de ne pas faire suivre les messages. Ceci leur permet d'économiser leur bande passante, mais ceci met aussi en danger le réseau car tout le trafic doit alors passer par les nœuds non-rationnels. Ces derniers peuvent alors se retrouver en trop faible proportion pour gérer tout le trafic ou pour assurer l'anonymat. Quelques travaux ont été effectués pour travailler sur le problème des nœuds rationnels [61, 62]. Cependant ils s'appuient toujours sur l'existence de nœuds suivant le protocole par altruisme, ce qui est une hypothèse forte.

3.3 $\mathcal{P}^5$

$\mathcal{P}^5$ [63] fait partie des rares protocoles à ne s'appuyer ni sur *Onion routing* ni sur DC-Net pour assurer son anonymat. $\mathcal{P}^5$ propose d'assurer l'anonymat grâce à la

multidiffusion des messages. L'idée est la suivante : si l'algorithme de multidiffusion est tel que chaque nœud participe en faisant suivre les messages envoyés par d'autres nœuds, il devient difficile pour l'adversaire de savoir si un nœud est la source d'un message ou s'il n'est qu'un relais. Ce mécanisme n'étant pas suffisant lorsque le trafic est faible, $\mathcal{P}^5$ demande aux nœuds de multidiffuser du faux trafic afin de masquer le trafic réel.

Afin d'effectuer la multidiffusion de façon efficace, $\mathcal{P}^5$ place les nœuds dans une structure d'arbre binaire. Cette structure permet ensuite de définir des groupes de multidiffusion comme illustré en figure 3.5 : un groupe de multidiffusion est constitué d'un nœud, le nœud C dans la figure, de l'arbre, de tous ses descendants (nœuds en dessous de lui dans l'arbre, A et B dans la figure) ainsi que de tous ses ascendants (tous les nœuds ayant n comme descendant, D et E dans la figure). Chaque nœud de l'arbre définit un groupe de multidiffusion. Pour envoyer un message, un nœud chiffre le message avec la clé publique du destinataire et diffuse le message chiffré dans un groupe de diffusion auquel appartiennent l'émetteur et le destinataire. Tous les nœuds appartiennent au groupe de multidiffusion défini par la racine de l'arbre. Il est donc toujours possible de faire parvenir un message à sa destination. L'inconvénient d'utiliser le groupe de multidiffusion défini par la racine de l'arbre est qu'il risque d'être retardé ou perdu. En effet, le nœud à la racine de l'arbre fait partie de tous les groupes de multidiffusion ; il reçoit donc tous les messages multidiffusés dans le système et risque donc d'être saturé de messages à faire suivre. Pour éviter ce problème, $\mathcal{P}^5$ propose un mécanisme permettant de faire suivre les messages d'un groupe de multidiffusion à un autre : à son arrivée, chaque nœud joint plusieurs points de l'arbre binaire. Pour chacun de ces points, le nœud indique, aux groupes de multidiffusion correspondants, la liste des autres groupes que le nœud a joint. Lorsqu'un nœud veut faire parvenir un message à un groupe de multidiffusion dont il ne fait pas partie, il chiffre le message avec la clé publique du destinataire, puis le chiffre avec la clé publique d'un nœud présent dans son groupe et celui du destinataire. Pour finir, il multidiffuse le message. À la réception du message, le nœud relais le déchiffre et le multidiffuse dans le groupe du destinataire.

L'une des propositions phares de $\mathcal{P}^5$ est de donner à chaque nœud la possibilité de choisir le niveau d'anonymat qu'il veut avoir. Ce choix se fait en choisissant quel groupe de multidiffusion utiliser pour envoyer des messages et à quels groupes accepter de répondre. Plus les groupes contiennent de nœuds, plus l'anonymat est fort. En contrepartie d'un anonymat plus fort, le nœud accepte une perte de performance : plus les groupes contiennent de nœuds, plus il y a de messages à faire suivre, plus il y a de risques que les nœuds soient surchargés et retardent ou abandonnent les messages qu'ils doivent faire suivre.

$\mathcal{P}^5$ assure l'anonymat des destinataires grâce à la multidiffusion des messages : si un nœud ne peut lier la clé avec laquelle il chiffre un message à un nœud, il ne peut savoir quel nœud, dans le groupe de multidiffusion, est capable de déchiffrer le message. Assurer l'anonymat des destinataires assure l'anonymat des relations. Comme nous l'avons dit précédemment, $\mathcal{P}^5$ essaye aussi d'assurer l'anonymat des émetteurs en s'appuyant sur la structure de multidiffusion et sur l'envoi de faux messages. Cependant, l'adversaire est capable de récolter des informations sur les messages reçus par les nœuds qu'il contrôle. Il peut notamment retracer le chemin suivi par le message en identifiant les nœuds sous son contrôle qui ont reçu le message, l'ordre dans lequel ils ont reçu le message et le voisin leur ayant envoyé le message. Le recoupement de ces

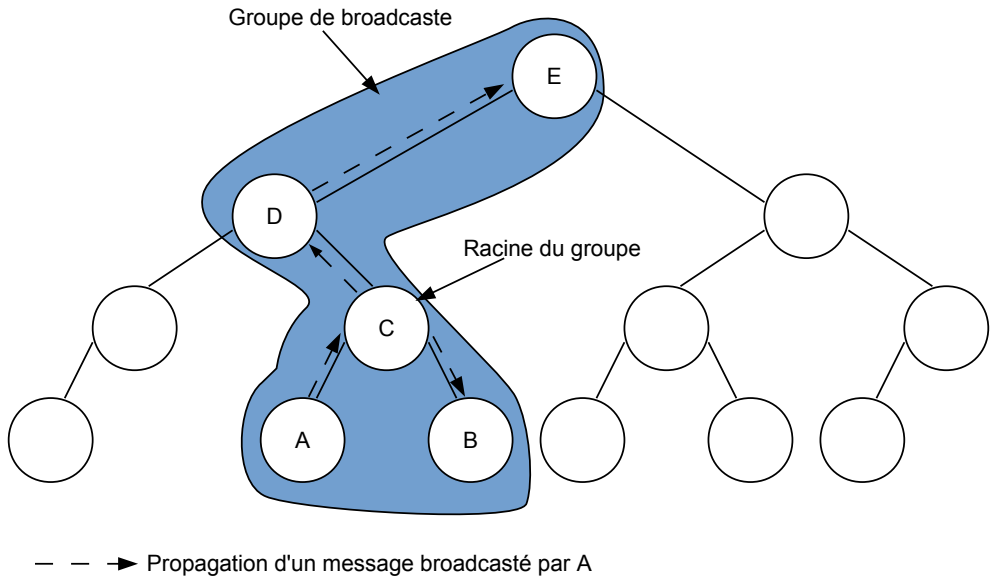


FIGURE 3.5 – Architecture du système $\mathcal{P}^5$

trois informations permet de fortement réduire la portion de l'arbre dans laquelle a été émis le message et ainsi de réduire, voire de casser l'anonymat de l'émetteur. Il est possible de réduire ce problème en utilisant systématiquement le système permettant de transmettre les messages d'un groupe de diffusion à un autre. De cette façon les nœuds servant de relais entre les groupes deviennent des mixs et l'adversaire n'est plus capable de remonter le chemin suivi par un message plus loin que ce "mix".

En plus de ne pas garantir l'anonymat de l'émetteur, $\mathcal{P}^5$ est victime de faiblesses dues à sa structure : les nœuds sont placés dans l'arbre en fonction de leur arrivée dans le système. L'adversaire peut donc choisir le moment où ses nœuds vont joindre le réseau de façon à les placer stratégiquement. L'adversaire peut par exemple couper l'arbre en deux et empêcher tout nœud se trouvant dans le bas de l'arbre de communiquer avec un nœud présent dans le haut de l'arbre.

Pour finir, $\mathcal{P}^5$ indique que les nœuds peuvent ignorer des messages lorsqu'ils sont surchargés. Cela signifie que tout nœud peut ignorer des messages. il est donc facile de construire un nœud rationnel qui ignore des messages pour économiser de la bande passante ou pour donner la priorité à ses propres messages. Vu que les messages se propagent en suivant les branches de l'arbre, un nœud rationnel à la racine d'un groupe de multidiffusion empêchera les messages venant d'une moitié du groupe de multidiffusion d'être reçu par les nœuds se trouvant dans l'autre moitié du groupe. Dans l'exemple de la figure 3.5, si C est rationnel il empêchera les messages de A d'atteindre les nœuds B, D et E. il suffit donc de peu de nœuds rationnels pour que $\mathcal{P}^5$ ne fonctionne plus.

3.4 Conclusion

Comme nous l'avons vu dans les sections précédentes, il existe un grand nombre de protocoles de communications anonymes. Cependant, aucun d'entre eux ne remplit l'ensemble des caractéristiques que l'on peut aujourd'hui attendre d'un protocole de communications anonymes. Un grand nombre des protocoles s'appuient sur un modèle client-serveur qui est à la fois une faiblesse d'un point de vue anonymat et une limite à la montée en charge du système. Une faiblesse, car un petit nombre de serveurs est facile à attaquer. Une limite à la montée en charge, car tout le trafic doit passer par les serveurs, ce qui en fait un goulot d'étranglement. Même Tor, qui possède une base de serveurs relativement large, grâce à sa facilité de déploiement, souffre d'un problème de débit lorsqu'un petit nombre de clients consomme excessivement la bande passante [62]. Pour leur part, les protocoles se basant sur le modèle pair-à-pair ne gèrent pas le problème des nœuds rationnels. Le seul protocole gérant les nœuds rationnel est Dissent v1 mais il ne fonctionne pas avec plus de 40 nœuds.

En conclusion, il faut un nouveau protocole pair-à-pair assurant l'anonymat des relations, des expéditeurs et des destinataires, tout en étant capable de monter en charge de gérer les nœuds rationnels.

4

Le protocole *RAC*

Sommaire

Dans ce chapitre nous présentons le protocole *RAC*. *RAC* est un protocole de communications anonymes assurant l'anonymat des expéditeurs, des destinataires et des communications. Il est de plus capable de monter en charge même en présence de nœuds rationnels. Afin de pouvoir monter en charge le protocole doit avoir un coût indépendant du nombre de nœuds dans le système. Pour ce faire nous nous appuyons sur deux idées principales : (1) un nombre de diffusions réduit et indépendant du nombre N de nœuds dans le système, et (2) des groupes de diffusion de taille réduite et indépendante de N. Nous expliquons, dans un premier temps, les mécanismes permettant de mettre en place ces deux idées. Dans un deuxième temps, nous donnons une description détaillée du protocole.

4.1 Idée clé numéro 1 : un nombre réduit de diffusions

Après avoir constaté que la principale limitation de Dissent v1 et Dissent v2 venait du nombre de diffusions impliquées par l'utilisation de DC-Net, nous avons décidé de nous baser sur les mécanismes d'*Onion routing* pour créer notre protocole. *Onion routing* est, en effet, le mécanisme de communications anonymes le plus efficace à ce

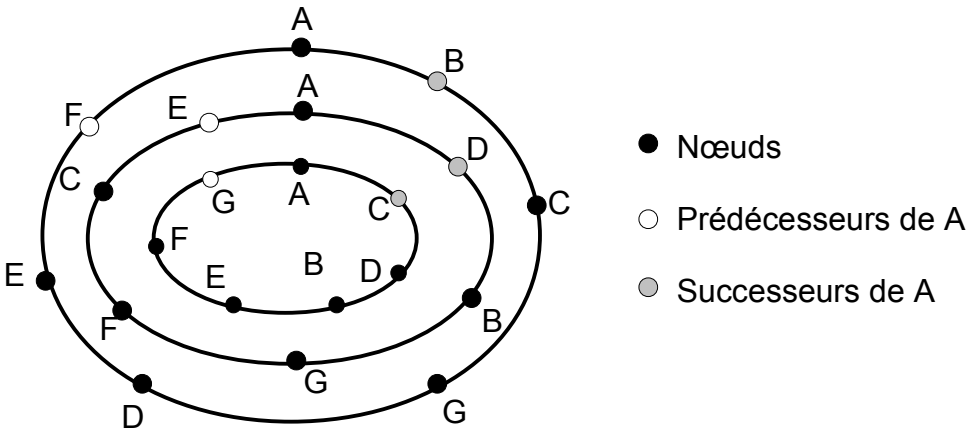


FIGURE 4.1 – La structure de Fireflies

jour. Comme décrit précédemment, le problème des protocoles pair-à-pair basés sur *Onion routing* est qu'ils ne prennent pas en compte les nœuds rationnels. Ces derniers peuvent donc décider de ne pas faire suivre certains messages. Il est donc nécessaire de trouver un moyen de surveiller le comportement des relais. La difficulté étant que seul l'expéditeur d'un message connait l'ensemble des relais par lesquels doit passer un oignon et sait à quoi doivent ressembler les différentes couches de l'oignon. C'est donc le seul nœud capable de surveiller le comportement des relais (pour cet oignon). Il nous faut donc trouver un moyen permettant à l'expéditeur d'un oignon de surveiller le comportement de ses relais sans révéler son identité. En effet, si un nœud sait qu'il est surveillé par un autre nœud, il pourra en déduire qu'il est relai pour ce nœud et casser son anonymat. Notre solution est de diffuser l'ensemble des messages : les émetteurs comme les relais diffusent les messages qu'ils envoient ou font suivre. De cette façon les expéditeurs reçoivent les messages envoyés par les relais et peuvent, sans révéler leur identité, vérifier que ces derniers ont bien fait suivre les messages.

Pour qu'un expéditeur ne puisse accuser à tort un relais il faut que le mécanisme de diffusion soit fiable en présence de nœuds rationnels et de nœuds contrôlés par l'adversaire. La solution la plus simple est d'utiliser un mécanisme de diffusion basique dans lequel chaque nœud envoie directement ses messages à chaque autre nœud du système. Ce système étant inefficace, nous avons conçu un protocole de diffusion s'appuyant sur des relais et forçant les nœuds à relayer les messages diffusés. Nous nous sommes appuyé pour cela sur une structure de réseau similaire à celle utilisée dans le protocole Fireflies [64]. Les nœuds sont placés sur plusieurs anneaux virtuels à l'aide d'une fonction de hachage, le résultat est présenté figure 4.1. Les nœuds entourant un nœud sur chacun des anneaux sont ses prédécesseurs (en blanc dans la figure 4.1) et ses successeurs (en gris dans la figure 4.1). Le mécanisme de diffusion est présenté figure 4.2 et fonctionne de la façon suivante : lorsqu'un nœud veut diffuser un message il l'envoie à l'ensemble de ses successeurs. À chaque fois qu'un nœud reçoit un message de l'un de ses prédécesseurs il le fait suivre à l'ensemble de ses successeurs. Ainsi, un nœud s'attend à recevoir chaque message de l'ensemble de ses prédécesseurs. Si un prédécesseur ne transmet pas un message, le nœud considère ce prédécesseur comme

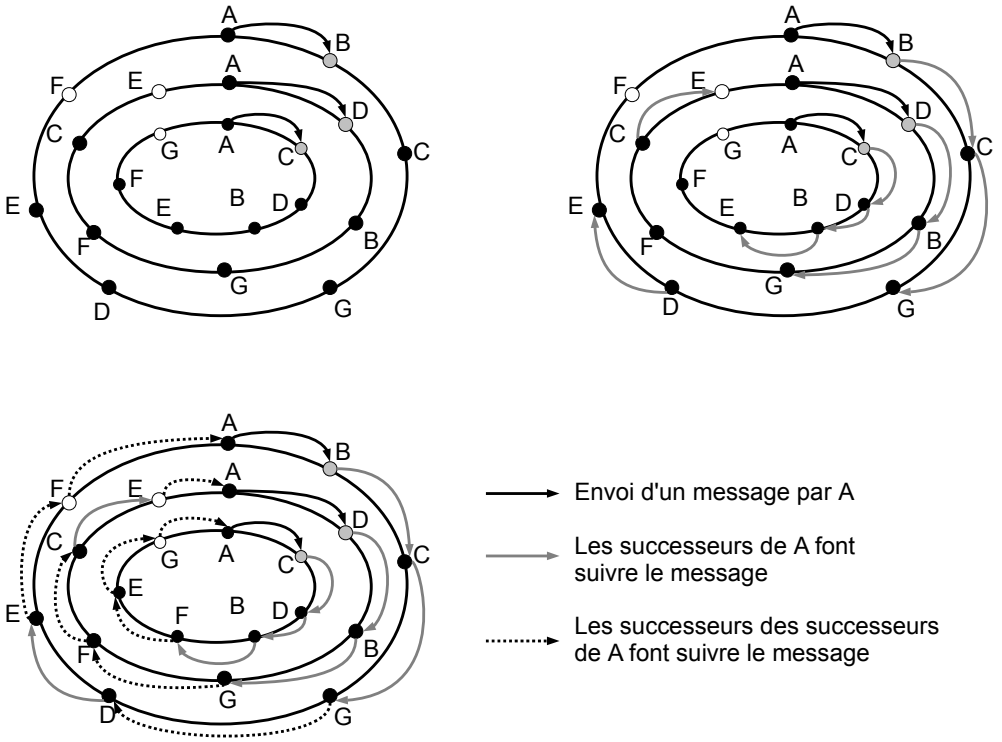


FIGURE 4.2 – Diffusion d'un message dans Fireflies

étant rationnel. Un nœud considéré comme rationnel par la majorité de ses successeurs est expulsé du système. Les nœuds sont ainsi forcés de faire suivre les messages qu'ils reçoivent. En plus de vérifier que leurs prédécesseurs font bien suivre les messages, les nœuds vérifient aussi qu'ils envoient des messages à débit constant. Comme nous le verrons dans la description détaillée du protocole, ce dernier point est nécessaire pour assurer l'anonymat.

La figure 4.3 illustre l'envoi d'un message à l'aide de *RAC*. Le nœud A veut envoyer un message au nœud D de façon anonyme. Il construit un oignon contenant ici les relais B et C. Le nœud A diffuse ensuite l'oignon en utilisant la structure d'anneaux présentée ci-dessus. A la réception d'un oignon chaque nœud commence par le faire suivre à ses successeurs, comme requis par le mécanisme de diffusion. Chaque nœud essaye ensuite de déchiffrer l'oignon. Lorsqu'un nœud parvient à déchiffrer l'oignon et obtient un nouvel oignon, il est relais et diffuse l'oignon résultant du déchiffrement. C'est le cas des nœuds B et C dans la figure. Lorsqu'un nœud parvient à déchiffrer l'oignon et obtient un message clair, il est destinataire du message. C'est le cas du nœud D dans notre exemple. Dans ce cas le nœud ne fait rien de plus que de participer à la diffusion en faisant suivre l'oignon reçu.

Le protocole décrit précédemment a un coût de $L \times R \times Bcast(N)$, avec $L << N$ le nombre de relais des oignons et $R << N$ le nombre d'anneaux du mécanisme de diffusion. Nous avons donc atteint notre premier objectif : rendre le nombre de diffusions nécessaires à l'envoi d'un message indépendant de N. Dans la section suivante nous expliquons comment nous rendons la taille des groupes de diffusions

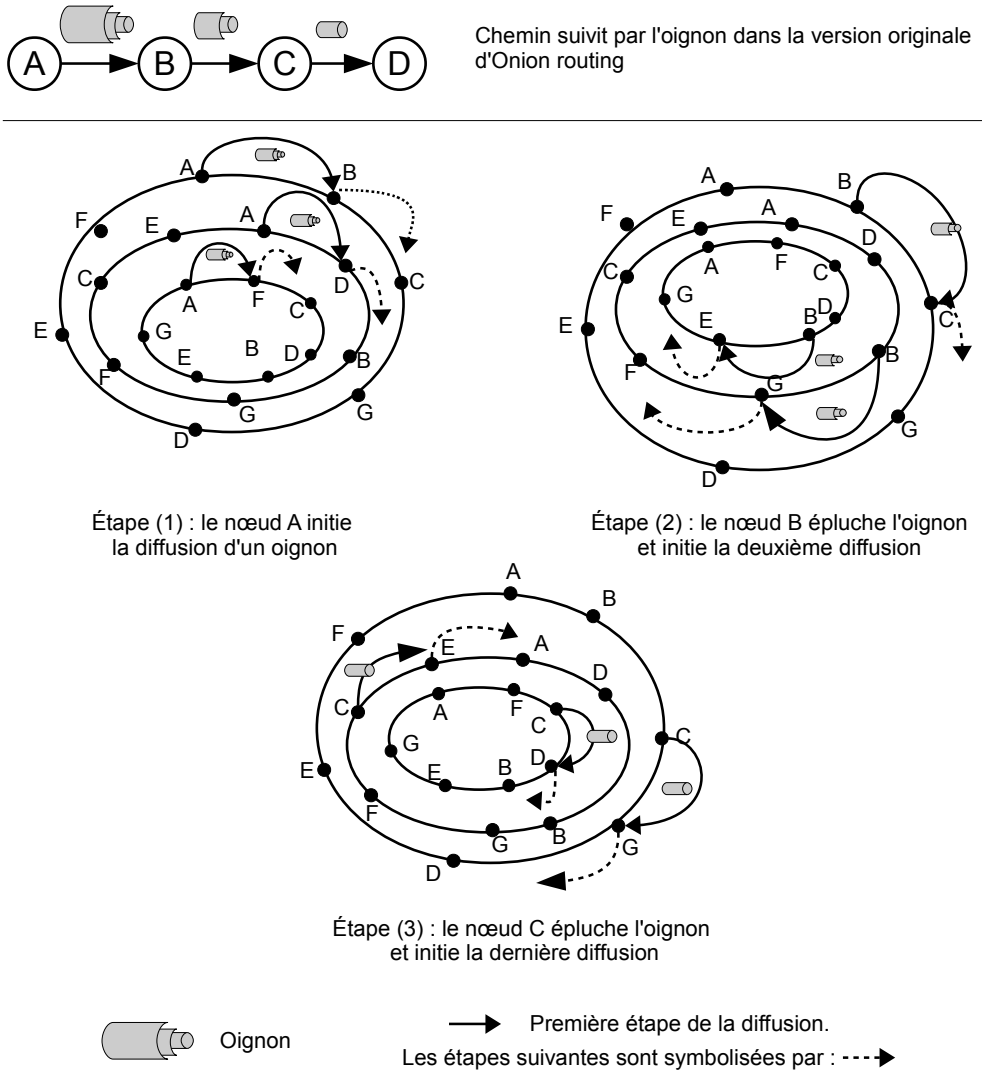


FIGURE 4.3 – Envoi d'un message dans *RAC*

indépendante de N.

4.2 Idée clé numéro 2 : des groupes de diffusion de taille réduite

Afin de rendre la taille des groupes dans lesquels les messages sont diffusés indépendante de N, nous proposons de diviser l'ensemble des nœuds en groupes de taille G. Il est à noter qu'utiliser des groupes de taille G au lieu de N permet à l'adversaire d'obtenir plus d'informations sur les possibles émetteurs et récepteurs des messages. Plus précisément, l'adversaire sait que l'émetteur d'un message est l'un des G nœuds du groupe dans lequel le message a été diffusé. De même pour le récepteur. L'anonymat des nœuds est donc de un parmi G au lieu de un parmi N. La taille de G étant un paramètre

du système, les utilisateurs devront faire un choix entre anonymat et performance au moment de la configuration de *RAC*.

Afin de créer les groupes et d'éviter que l'adversaire ne puisse s'attaquer à un groupe en particulier, l'identifiant du groupe auquel un nœud doit appartenir est calculé de façon déterministe lorsque le nœud joint le système. Par exemple en utilisant le hash de sa clé publique modulo le nombre de groupes. Lorsque deux nœuds appartenant au même groupe veulent communiquer, ils utilisent le protocole décrit dans la section précédente au sein de leur groupe. De cette façon, le coût de l'envoi d'un message est de $L \times R \times Bcast(G)$. La taille des groupes étant fixe le coût est indépendant de N. La taille des groupes étant fixe et indépendante de N, il y a une forte probabilité pour que deux nœuds voulant communiquer ne se trouvent pas dans le même groupe. Comment les fait-on alors communiquer ? La première idée est de faire des super-groupes regroupant les groupes deux par deux et d'y utiliser le protocole décrit dans la section précédente. Cette solution aurait alors un coût de $L \times R \times Bcast(2G)$. Mais une simple optimisation, que nous allons décrire ci-dessous, permet de réduire ce coût.

Pour envoyer un message à un nœud ne se trouvant pas dans son groupe, un émetteur envoie son message, en utilisant le protocole de la section précédente, comme si le destinataire se trouvait dans son groupe. Il ajoute cependant un marqueur à l'intérieur de la couche d'oignon qui sera déchiffrée par le dernier relais. Ce marqueur indique au dernier relais le groupe dans lequel se trouve le destinataire. De cette façon les diffusions nécessaires au $L - 1$ premier relais sont faites au sein du groupe de l'expéditeur et ont un coût de $Bcast(G)$. Lorsqu'il lit le marqueur le dernier relais doit transmettre l'oignon au groupe du destinataire tout en informant l'expéditeur qu'il a bien fait suivre le message. Afin d'effectuer ces deux actions, le dernier relais diffuse l'oignon à un super-groupe contenant l'ensemble des nœuds du groupe de l'expéditeur et du groupe du destinataire. Dans le reste du document, nous appelons ces super-groupes des *chaînes*. Grace à cette optimisation le coût du protocole est de $(L-1) \times R \times Bcast(G)$, pour les messages envoyés au sein du groupe de l'expéditeur, plus $R \times Bcast(2G)$, pour l'envoi du message dans la chaîne. Comme $Bcast(2G) = 2 \times Bcast(G)$ le coût total de notre protocole est $(L + 1) \times R \times Bcast(G)$ qui est inférieur à $L \times R \times Bcast(2G) = 2 \times L \times R \times Bcast(G)$, car $L + 1 \leq 2 \times L$.

Nous proposons donc, grâce à ces deux idées, un protocole d'anonymat fonctionnant en présence de nœuds rationnels et dont le coût est indépendant du nombre de nœuds dans le système.

4.3 Description détaillée du protocole

Dans cette section nous décrivons en détails le protocole *RAC*.

4.3.1 Joindre le système

Chaque nœud présent dans le système possède un identifiant (ID), une vue contenant la liste des nœuds présents dans le système et deux jeux de clés publique/privée. La première paire est liée à l'identifiant du nœud et est utilisée lors de la construction des oignons. Nous appelons ce jeu de clés "les clés ID". La seconde paire est utilisée pour chiffrer chaque message afin qu'il ne soit déchiffrable que par son destinataire.

Ce jeu de clés ne peut être lié à l'identifiant du nœud. Ceci permet d'assurer l'anonymat des destinataires. Nous appelons ce jeu de clés "les clés pseudonymes". La façon dont les nœuds découvrent la clé pseudonyme publique du nœud avec lequel ils veulent communiquer dépend de l'application. Par exemple, dans un système de publication/souscription anonyme, les nœuds souscriraient aux sujets qui les intéressent en utilisant leur clé pseudonyme publique.

La position d'un nœud dans le système est déterminée par son identifiant. Afin d'éviter qu'un opposant ne puisse joindre le groupe de son choix, le calcul de l'identifiant est fait selon une méthode inspirée d'Herbivore [6]. Le calcule fonctionne comme suit. Soit f et g deux fonctions à sens unique [1]. Soit K la clé ID publique du nouvel arrivant. Le nouvel arrivant génère des vecteurs aléatoires jusqu'à ce qu'il obtienne un vecteur $y \neq K$ tel que les mk premiers bits de $f(K)$ soient égaux aux mk premiers bits de $f(y)$. La valeur de $g(K, y)$ donne au nouvel arrivant son identifiant. Cette solution évite qu'un opposant puisse choisir le groupe qu'il va joindre, car il est très difficile d'obtenir les valeurs de K et y permettant d'obtenir une valeur voulue pour $g(K, y)$ lorsque g et f sont des fonctions à sens unique.

Pour joindre le système un nœud n envoie une requête $JOIN$ à un nœud x déjà présent dans le système. Cette requête contient la clé ID publique de n, appelée K, l'identifiant de n, noté ID, et le vecteur y. Lorsque x a reçu la requête $JOIN$ il détermine le groupe que n doit joindre. Par exemple le groupe contenant le nœud avec l'ID le plus proche. x diffuse, ensuite, de façon anonyme, la requête $JOIN$ au groupe que n doit joindre. Lorsqu'ils reçoivent la requête, les nœuds du groupe vérifient que l'ID de n est correct. c'est-à-dire que $y \neq K$, que les mk premiers bits de $f(K)$ sont égaux aux mk premiers bits de $f(y)$, que $ID = g(K, y)$ et qu'il n'y a pas déjà un autre nœud ayant joint le système avec les mêmes valeurs de K et y. Ce dernier point permet d'éviter qu'un opposant essaye de rejoindre plusieurs fois le même groupe en utilisant les mêmes clés et ID. Si l'ID n'est pas correct, la requête est ignorée ; sinon, les nœuds ajoutent n à leur vue et calculent la position de n sur les différents anneaux du système de diffusion. Ce calcul est effectué de façon identique au calcul effectué dans le protocole Fireflies [64] : la position d'un nœud sur le i^{me} anneau est déterminée par le hash du couple (ID, i). Le nombre d'anneaux dépend de la taille du système et du nombre d'opposants à tolérer. Plus précisément, l'ensemble des successeurs d'un nœud (les successeurs directs du nœud sur chacun des anneaux) doit contenir une majorité de nœuds n'étant pas contrôlés par l'opposant. De plus cette majorité doit être assez importante pour que l'opposant ne puisse pas perturber le mécanisme de diffusion. Cela signifie que, comme prouvé par A.-M. Kermarrec & al [65], chaque nœud doit avoir au moins $log(N) + c$ successeurs qui ne sont pas des opposants, avec c une constant et N le nombre de nœuds dans le système. En pratique, dans un système contenant 1000 nœuds et avec 10% de nœuds contrôlés par l'opposant, il suffit de 7 anneaux pour assurer que l'ensemble des successeurs de chaque nœud contienne moins de 3 opposants avec une probabilité de 0.999.

Après un temps T correspondant au temps maximum nécessaire pour que la requête $JOIN$ parvienne à tous les nœuds du groupe de destination, le nœud x envoie à n un message $PRES$ contenant la liste des nœuds présent dans le groupe que n doit rejoindre. Ce message informe n que tous les nœuds du groupe ont été informés de son

1. Fonction qui peut être aisément calculée, mais qui est difficile à inverser (ex : fonction de hachage)

arrivée dans le système. Lorsque n reçoit le message $PRES$, il calcule sa position sur les anneaux et envoie un message à l'ensemble de ses successeurs et prédécesseurs afin de leur indiquer qu'ils peuvent maintenant l'utiliser comme successeur ou prédécesseur. Les nœuds du groupe attendent une période de $2T$ après la réception de la requête $JOIN$ avant d'utiliser n comme un relais. Cette période d'attente permet aux nœuds de s'assurer que n a bien joint le système avant de l'utiliser comme relais. Une fois que n a joint son groupe, les nœuds de ce groupe diffusent la requête $JOIN$ dans les chaînes englobant le groupe. De cette façon, n peut trouver sa position dans chacune des chaînes et tous les nœuds du système sont informés de l'arrivée de n.

4.3.2 Gestion des groupes

Pour assurer une borne inférieure à l'anonymat et une borne supérieure au coût du protocole, les groupes doivent avoir une taille supérieure à t_{min} et inférieure à t_{max}. Ces deux valeurs sont des paramètres du système. Lorsque la taille d'un groupe devient inférieure à t_{min}, les nœuds appartenant à ce groupe diffusent un message indiquant que le groupe doit être dissout. Ils réeffectuent ensuite la procédure de jointure du système afin d'être assigné à un autre groupe. Lorsqu'un nœud joint le système et qu'un groupe devient plus grand que t_{max}, les nœuds de ce groupe diffusent un message indiquant que le groupe doit être divisé en deux. Les nœuds calculent ensuite lequel des deux nouveaux groupes ils doivent joindre. Par exemple, les nœuds avec les IDs les plus bas joignent le premier groupe et les nœuds avec les IDs les plus hauts joignent le deuxième groupe. Pour finir, chaque nœud calcule sa position sur les anneaux de son groupe et se connecte à ses nouveaux successeurs et prédécesseurs. Il est à noter que pour que ce mécanisme fonctionne il faut $t_{min} \leq \frac{t_{max}}{2}$.

4.3.3 Envoi d'un message

Pour envoyer un message, un nœud commence par chiffrer ce message à l'aide de la clé pseudonyme publique du destinataire. Le nœud choisit ensuite aléatoirement L clés ID publiques des membres de son groupe et chiffre le message en couches successives. Il forme ainsi un oignon. Chaque couche contient un marqueur qui permet aux relais de savoir qu'ils ont bien déchiffré une couche. Si le destinataire du message se trouve dans une autre chaîne, le marqueur contenu dans la couche correspondant au dernier relais contient les informations sur la chaîne dans laquelle le message doit être envoyé. Une fois que les L couches ont été créées, l'émetteur enroberre le message, afin d'obtenir la taille de message prédéfinie par le système, et démarre la diffusion en envoyant le message à chacun de ses successeurs. Les messages sont enrobés pour qu'il soit impossible à un opposant de suivre un message en se basant sur la taille des blocs échangés sur le réseau.

Il est à noter qu'afin de garantir leur anonymat, les nœuds diffusent des messages à débit constant. Ces messages peuvent être leur propres messages, des oignons à faire suivre ou de faux messages, appelé bruit, lorsqu'ils n'ont pas de messages à envoyer. Le bruit oblige les nœuds à payer un surcout en bande passante lorsqu'ils n'ont pas de messages à envoyer, mais ceci garantit en contrepartie qu'un adversaire ne peut pas se baser sur la fréquence des envois pour déterminer si un nœud envoie réellement des messages.

4.3.4 Réception d'un message

Lorsqu'un nœud reçoit un message il vérifie qu'il ne l'a pas déjà reçu. S'il n'a pas déjà reçu le message le nœud fait suivre ce message à ses successeurs sur les anneaux de la chaîne ou du groupe dans lequel est diffusé le message. Le nœud vérifie ensuite s'il peut déchiffrer le message à l'aide de sa clé ID privée. Si le nœud parvient à déchiffrer le message (il peut lire le marqueur indiquant le bon déchiffrement), cela signifie que le nœud doit agir comme un relais pour ce message. Il vérifie alors si le marqueur indique une chaîne dans laquelle faire suivre l'oignon, enrobe l'oignon et envoie l'oignon à ses successeurs sur les anneaux du groupe ou de la chaîne appropriée. S'il ne parvient pas à déchiffrer le message avec sa clé ID privée, le nœud essaye de déchiffrer le message à l'aide de sa clé pseudonyme privée. S'il parvient à déchiffrer le message, il est destinataire du message et le livre à son utilisateur. Il est à noter que seule la couche interne des oignons est diffusée sur les chaînes. Les nœuds peuvent donc se contenter d'utiliser leurs clés pseudonymes privées pour essayer de déchiffrer les messages reçus sur les chaînes.

4.3.5 Vérification du comportement des nœuds

Afin d'éviter les agissements rationnels, les nœuds vérifient que (1) les relais qu'ils utilisent pour envoyer leurs messages font correctement suivre les messages, (2) les nœuds qui les précédent sur les anneaux des groupes et chaînes font correctement suivre les messages diffusés (une et une seule fois), et (3) les nœuds qui les précédent sur les anneaux de leur groupe envoient des messages à un débit constant. Chaque fois qu'un nœud découvre un mauvais comportement il met le nœud rationnel sur une liste noire. Chaque nœud maintient plusieurs listes noires : une liste noire pour les prédécesseurs de leur groupe qui ne font pas suivre certains messages ou n'envoient pas de messages à un débit constant, une liste noire par chaîne pour les prédécesseurs qui ne font pas suivre certains messages et une liste noire pour les relais qui ne font pas suivre les messages. Nous expliquons dans la section suivante comment les listes noires sont utilisées pour expulser les nœuds.

La vérification (1) est effectuée comme suit. Lorsqu'un nœud envoie un message il garde une copie de chacune des couches de l'oignon qu'il crée. Il associe chacune de ces couches à l'identifiant du relais qui doit la diffuser. Ainsi, le message final est associé à l'identifiant du dernier relais, la dernière couche de l'oignon est associée à l'identifiant de l'avant dernier relais, etc. Le nœud s'attend ensuite à recevoir les messages correspondant aux différentes couches avant l'expiration d'un compte à rebours (car tous les messages sont diffusés). Le premier relais, s'il existe, qui ne diffuse pas correctement sa version de l'oignon est mis sur la liste noire des relais et n'est plus utilisé comme relais par l'expéditeur.

La deuxième vérification est effectuée comme suit. Étant donné que les messages sont diffusés, les nœuds s'attendent, pour chaque message, à recevoir une copie par prédécesseur sur les anneaux de la chaîne ou sur les anneaux du groupe dans lequel le message est diffusé. Les nœuds démarrent donc un compte à rebours à la première réception d'un message et si un prédécesseur n'envoie pas de copie dans le temps imparti il sera suspecté par son successeur qui l'ajoutera à la liste noire appropriée. Un nœud envoyant un message deux fois sera aussi suspecté et mis en liste noire. La

vérification du double envoi de messages est nécessaire pour éviter les attaques par répétition [66].

La troisième vérification est effectuée comme suit. Chaque nœud vérifie qu'il reçoit à débit constant des messages provenant de ses prédécesseurs sur chacun des anneaux de son groupe. Si ce n'est pas le cas, le nœud ajoute son prédécesseur à la liste noire des prédécesseurs.

4.3.6 Expulsion de nœuds

Comme décrit dans la section précédente, les nœuds maintiennent deux genres de listes noires : la liste noire des relais, et les listes noires des prédécesseurs. Les nœuds partagent ces listes noires de la façon suivante. Chaque nœud envoie périodiquement sa liste noire des relais aux nœuds de son groupe. Cette liste peut donner des informations propres à casser l'anonymat des messages émis. Pour cette raison, la liste des relais doit être envoyée de façon anonyme. Pour éviter une attaque lors de laquelle des nœuds contrôlés par l'opposant enverraient plusieurs listes noires par période, l'envoi anonyme se fait à l'aide du protocole de mélange vérifiable de Dissent v1. Ce protocole permet en effet de mélanger un groupe de messages de taille donnée et de diffuser ces messages avec un anonymat cryptographiquement fort. Les listes noires des prédécesseurs sont diffusées en clair aux nœuds de la chaîne ou du groupe auquel elles correspondent.

Un nœud A supprime un nœud B de sa vue dès qu'il a reçu les preuves que : (1) B appartient à la liste noire de $(t+1)$ de ses successeurs dans une chaîne ou un groupe, avec t le nombre maximum de successeurs contrôlés par l'opposant qu'un nœud peut avoir (comme définit dans Fireflies [64]) ; ou (2) B appartient à la liste noire des relais de $(f+1)$ nœuds de son groupe, avec f le nombre maximum de nœuds contrôlés par l'opposant dans un groupe. f est une variable du système : l'utilisateur définit le nombre maximum de nœuds que l'opposant peut contrôler et garanti l'anonymat du système tant que le nombre de nœuds contrôlés par l'opposant est inférieur à ce nombre. Nous évaluons l'influence de la valeur de f sur l'anonymat garantit par le système dans le chapitre 6. Une fois le nœud enlevé de sa vue, A vérifie que la liste de ses prédécesseurs et la liste de ses successeurs n'ont pas changé. Si l'une de ces listes a changé, A se connecte à son nouveau prédécesseur ou successeur.

Lorsqu'un nœud est expulsé d'un groupe ou d'une chaîne, les nœuds de ce groupe ou de cette chaîne doivent diffuser un message à l'ensemble des chaînes pour informer tous les nœuds de l'expulsion de ce nœud. Les nœuds ne diffusant pas ce message sont suspectés par les nœuds de leur groupe. Les nœuds d'une chaîne attendent d'avoir reçu $f+1$ notifications d'expulsion avant de prendre en compte cette expulsion.

4.4 Conclusion

Nous avons présenté le protocole *RAC*, un protocole de communications anonymes assurant l'anonymat des expéditeurs, des destinataires et des communications et capable de monter en charge en présence de nœuds rationnels. Nous avons montré comment l'utilisation d'*Onion routing*, là où les autres protocoles utilisent DC-Net, permet à *RAC* de fonctioner avec un nombre réduit de diffusion. Nous avons ensuite montré comment la séparation des nœuds en groupes permetait à *RAC* d'avoir un coût indépendant du

nombre de nœuds dans le système. Pour finir nous avons détaillé le fonctionement du protocole *RAC*.

Dans les chapitres suivants nous commencerons par prouver, dans un premier chapitre, que le protocole *RAC* garantit l'anonymat des expéditeurs, des destinataires et des communications et que le protocole fonctionne bien en présence de nœuds rationnels. Nous évaluerons ensuite les performances du protocole dans un deuxième chapitre.

5

Preuves

Sommaire

Dans ce chapitre nous commençons par prouver que *RAC* assure l'anonymat des expéditeurs, l'anonymat des destinataires et l'anonymat des relations. Nous prouvons ensuite que *RAC* gère correctement les nœuds rationnels.

5.1 Preuves d'anonymat

Dans cette section nous prouvons que *RAC* assure l'anonymat des expéditeurs, l'anonymat des destinataires et l'anonymat des relations. Pour ce faire nous utilisons les notations suivantes : le nombre de nœuds est noté N, la taille des groupes est notée G, la proportion de nœuds contrôlés par l'adversaire est notée f et le nombre de relais utilisés lors de l'envoi anonyme de messages est noté L. Nous avons expliqué en section 4.2 que séparer les nœuds en groupes diminuait l'anonymat de "un parmi N" à "un parmi G". Nous allons maintenant montrer comment l'anonymat de un parmi G est protégé. Dans un premier temps nous allons prouver les garanties d'anonymat en présence d'un adversaire passif. Cela signifie que les nœuds contrôlés par l'adversaire suivent le protocole. L'adversaire ne peut donc obtenir des informations qu'en enregistrant ce qui passe sur le réseau. Nous étudierons le cas d'un adversaire actif dans un second temps.

5.1.1 Cas d'un adversaire passif

5.1.1.1 Anonymat des émetteurs

Les nœuds de *RAC* envoient des messages à débit constant, il n'est par conséquent pas possible de savoir s'ils sont en train d'envoyer un message, de faire suivre un message ou d'envoyer du bruit. De cette façon, si l'adversaire intercepte un message quelconque, il peut savoir qui a envoyé le message, mais il ne peut pas savoir si cet envoyeur est un relais ou un émetteur. Lorsque le destinataire est contrôlé par l'adversaire, ce dernier ne peut connaitre que l'identité du dernier relais. Grace au système d'oignon, il est garanti que si le dernier relais n'est pas contrôlé par l'adversaire, l'adversaire ne pourra pas connaitre l'identité de l'avant dernier relais. Il en va de même pour chacun des relais précédents. Ainsi, un nœud adverse destinataire d'un message ne peut trouver l'identité de son émetteur qu'en collaborant avec tous les relais de l'oignon. Ceci signifie que tous les relais sont contrôlés par l'adversaire, ce qui a une chance très faible de se produire. En effet, les relais étant sélectionnés de façon aléatoire par l'émetteur, il est peu probable que ce dernier ne sélectionne que des relais contrôlés par l'adversaire. Ainsi, la probabilité pour que l'émetteur ne sélectionne que des relais contrôlés par l'adversaire est égale à $\prod_{i=0}^{L} \frac{X-i}{G-i}$, avec X le nombre de nœuds adverses dans le groupe. Pour augmenter ses chances de succès, un adversaire essayant de casser l'anonymat d'un nœud cible doit donc contrôler un nombre X de nœuds, dans le groupe ciblé, aussi grand que possible. Comme les nœuds sont répartis de façon aléatoire dans les groupes, la probabilité que l'adversaire contrôle X nœuds dans un groupe donné est égale à $\prod_{i=0}^{X-1} \frac{fN-i}{N-i}$. La probabilité que l'adversaire casse l'anonymat d'un émetteur est donc au plus égale à $\max_X(\prod_{i=0}^{L} \frac{X-i}{G-i-1} \times \prod_{i=0}^{X-1} \frac{fN-i}{N-i})$. Avec $N = 100000$, $G = 1000$, $f = 5\%$ et $L = 5$ cette probabilité est de 5.7×10^{-25}, ce qui est extrêmement faible.

Il est à noter qu'il est plus compliqué pour l'adversaire d'essayer de faire des corrélations entre les messages reçus par un nœud et les messages émis par celui-ci dans *RAC* que dans *Onion routing*. En effet, dans *Onion routing* un nœud recevant un message est soit sa destination soit un relais, l'adversaire peut donc surveiller les réceptions et émissions d'un nœud pour essayer de faire des corrélations. Dans *RAC* les nœuds reçoivent tous les messages diffusés dans leur groupe. Un nœud recevant un message peut donc soit être un destinataire, soit être un relais, soit, plus probablement, n'avoir aucun rôle vis à vis de ce message. Un adversaire essayant de suivre le chemin suivi par un message doit donc essayer de faire des corrélations entre les messages reçus et les messages émis par l'ensemble des nœuds du groupe.

5.1.1.2 Anonymat des récepteurs

Lorsqu'un nœud veut envoyer un message m à un nœud n_D, il chiffre m avec la clé pseudonyme publique de n_D et diffuse le message à l'aide de relais. Comme expliqué en section 4.3.1 la clé publique de n_D ne peut pas être liée à n_D. Cette information ne peut donc pas être utilisée pour casser l'anonymat de n_D. Une fois le message m diffusé, seul n_D est capable de le déchiffrer à l'aide de sa clé pseudonyme privée. D'un point de vu extérieur, n_D se comporte comme tous les autres nœuds du groupe : il fait suivre l'oignon qu'il a reçu à chacun de ses successeurs une et une seule fois.

Par conséquent, il est impossible pour un adversaire de savoir que n_D est parvenu à déchiffrer le message m. Cet anonymat est optimal : la seule façon de casser l'anonymat des récepteurs est de contrôler tous les nœuds d'un groupe sauf un.

5.1.1.3 Anonymat des relations

Comme nous venons de le montrer, *RAC* assure un anonymat des récepteurs optimal. Par conséquent il est impossible pour un adversaire de déterminer la destination d'un message donné au sein d'un groupe. Il est ainsi impossible pour l'adversaire de déterminer si deux nœuds sont en cours de communication. Au mieux l'adversaire peut s'avoir qu'un nœud présent dans un groupe communique avec un nœud présent dans un autre groupe, lorsqu'un message est diffusé sur la chaîne liant ces deux groupes. Mais il ne peut pas savoir quel nœud est destinataire au sein du second groupe. *RAC* assure donc un anonymat des relations aussi fort que son anonymat des récepteurs.

5.1.2 Cas d'un adversaire actif

5.1.2.1 Anonymat des émetteurs

Un adversaire actif peut essayer de casser l'anonymat des émetteurs des façons suivantes : (1) en essayant de forcer un nœud à construire un oignon ne comportant que des relais contrôlés par l'adversaire ou (2) en essayant d'expulser certains nœuds du système. Expulser des nœuds du système peut permettre de casser l'anonymat de deux façons : premièrement cela réduit le nombre de nœuds dans le système n'étant pas contrôlés par l'adversaire et augmente mécaniquement la probabilité que les nœuds construisent des oignons ne contenant que des relais contrôlés par l'adversaire ; deuxièmement, cela peut permettre à l'adversaire d'effectuer une attaque par intersection [20] en comparant les messages envoyés dans le système avant et après l'expulsion d'un nœud.

Première attaque. Afin d'effectuer la première attaque, un nœud contrôlé par l'adversaire peut supprimer des messages qu'il devrait relayer. De cette façon il force l'émetteur à créer un nouvel oignon. À chaque fois que l'expéditeur crée un nouvel oignon il choisit aléatoirement de nouveaux relais et a donc une chance, très faible, mais non nulle de construire un oignon ne comprenant que des relais contrôlés par l'adversaire. Il est donc, hypothétiquement, possible de forcer un nœud à construire de nouveaux oignons jusqu'à ce qu'il construise un oignon uniquement composé de relais contrôlés par l'adversaire. Cependant, ceci n'est pas possible dans *RAC*. En effet, un nœud qui ne relaie pas les messages est mis sur la liste noire de l'expéditeur et n'est plus utilisé comme relais par celui-ci. Par conséquent, si le système contient une fraction f de nœuds adverses, si ces nœuds coordonnent leurs actions et si à chaque fois que l'expéditeur construit un oignon il contient un nœud contrôlé par l'adversaire, l'adversaire pourra forcer au plus fN création d'oignons. La probabilité que l'adversaire parvienne à forcer l'émetteur à créer un oignon ne contenant que des nœuds adversaire est donc au plus égale à fN fois la probabilité qu'un nœud construise un oignon ne contenant que des relais contrôlés par l'opposant. Nous avons évalué cette probabilité précédemment et celle-ci vaut $\max_X(\prod_{i=0}^{L} \frac{X-i}{G-i-1} \times \prod_{i=0}^{X-1} \frac{fN-i}{N-i})$, avec

X un entier compris entre 0 et N. La probabilité que l'adversaire parvienne à forcer l'émetteur à créer un oignon ne contenant que des nœuds adversaire est donc au plus égale : $fN \times \max_X(\prod_{i=0}^{L} \frac{X-i}{G-i-1} \times \prod_{i=0}^{X-1} \frac{fN-i}{N-i})$. Ainsi, dans un système tel que $n = 100000$, $G = 1000$, $f = 5\%$ et $L = 5$, la probabilité que l'adversaire parvienne à forcer un nœud à créer un oignon ne comportant que des relais contrôlés par l'adversaire est de 2.8×10^{-23}. Il faut, de plus, noter que nous présentons ici un cas théorique extrême, il est en effet, en pratique, impossible aux nœuds contrôlés par l'adversaire de coordonner leurs actions. Pour coordonner leurs actions, les nœuds contrôlés par l'adversaire devraient déterminer l'identité de l'émetteur de chaque message avant de décider s'ils le suppriment ou pas. Sans cela ils risquent de supprimer des messages émis par différents nœuds et de se faire expulser du système avant même d'avoir forcé un unique nœud à construire fN oignons. Savoir quel nœud est l'émetteur d'un oignon étant l'information que l'adversaire cherche à obtenir, la coordination est impossible.

Deuxième attaque. L'adversaire peut essayer de casser l'anonymat des émetteurs en faisant expulser des nœuds du système. Ceci lui permettrait par exemple d'effectuer une attaque par intersection [20]. Dans *RAC*, un nœud n est expulsé si : (1) $fG + 1$ nœuds de son groupe indiquent qu'il ne remplit pas ses fonctions de relais ou (2) la majorité des successeurs de n, dans son groupe ou dans une chaîne, indiquent que le nœud ne participe pas correctement à la diffusion. Considérons tout d'abord le premier cas. Il y a une proportion f de nœuds contrôlés par l'adversaire dans le système. Comme les nœuds sont placés dans les groupes de façon uniformément aléatoire, la proportion de nœuds contrôlés par l'adversaire dans chaque groupe est aussi f. Il y a donc fG nœuds contrôlés par l'adversaire dans chaque groupe. Par conséquent, l'adversaire ne peut pas provoquer l'expulsion d'un nœud en le mettant sur les listes noires des nœuds sous son contrôle. Pour faire expulser le nœud n l'adversaire doit parvenir à faire croire à un nœud d qu'il ne contrôle pas que n ne suit pas le protocole. De cette façon l'adversaire obtiendra un vote d'un nœud non-adversaire, vote auquel il pourra ajouter ses fG votes pour obtenir l'expulsion du nœud. L'unique façon dont l'adversaire peut faire mettre n sur la liste noire de d est d'empêcher certain messages d'atteindre n. L'adversaire peut alors espérer que n soit relais sur l'un des messages qu'il ne recevra pas. n sera alors mis sur liste noire pour ne pas avoir relayé un oignon. Cependant, comme montré dans Fireflies [64], il est possible d'améliorer la fiabilité du mécanisme de diffusion en augmentant le nombre d'anneaux. Ainsi, il est possible de choisir un nombre d'anneaux tel que l'adversaire ait une probabilité aussi faible que souhaitée de parvenir à empêcher un message d'atteindre n. Il est donc possible de s'assurer que l'adversaire ait une probabilité aussi faible que souhaité de parvenir à faire blacklister n par d. Considérons maintenant le second cas. Grace à la structure de Fireflies, plus le nombre d'anneaux est important plus la probabilité que n ait une majorité de successeurs contrôlés par l'adversaire est faible. Il est donc possible de choisir un nombre d'anneaux tel que l'adversaire ait une probabilité aussi faible que souhaité de pouvoir faire expulser un nœud grâce au contrôle d'une majorité de successeurs. Par exemple, avec $f = 5\%$, un nombre d'anneaux égale à 7 garantit que chaque nœud a une probabilité inférieure à 6×10^{-6} d'avoir une majorité de successeurs contrôlés par l'adversaire. Pour forcer l'éviction de n, l'adversaire doit donc parvenir à faire mettre n sur la liste noire d'un ou de plusieurs de ses successeurs non contrôlés par l'adversaire. Mais, pour les mêmes

raisons que dans le premier cas d'expulsion, ceci n'est pas possible si le nombre d'anneaux est assez élevé.

5.1.3 Anonymat des destinataires

Comme montré dans le cas d'un adversaire passif, l'anonymat des récepteurs ne peut être cassé que si l'adversaire contrôle tous les nœuds d'un groupe sauf un. L'adversaire peut donc essayer de casser l'anonymat des destinataires d'un nœud en faisant expulser du groupe tous les nœuds qu'il ne contrôle pas. Mais, comme nous l'avons vu dans la section sur l'anonymat des émetteurs, ceci n'est pas possible si le nombre d'anneaux est correctement choisi.

5.1.4 Anonymat des communications

Les deux sections précédentes, sur l'anonymat des émetteurs et l'anonymat des récepteurs, montrent qu'un adversaire actif ne peut parvenir à obtenir plus d'informations sur les nœuds et leurs liens aux messages qu'un adversaire passif. On peut donc montrer, de la même façon que dans le cas d'un adversaire passif, que l'anonymat des communications est protégé par le simple fait que l'anonymat des destinataires est optimal.

5.2 Tolérance aux nœuds rationnels

Dans cette section nous prouvons que *RAC* tolère la présence de nœuds rationnels, c'est-à-dire que les nœuds rationnels n'ont rien à gagner en déviant du protocole. Pour faire cette preuve, nous allons prouver que *RAC* est un équilibre de Nash [67]. Un équilibre de Nash est un concept de théorie des jeux dans lequel l'ensemble des choix fait par plusieurs joueurs est devenu stable du fait qu'aucun ne peut modifier seul sa stratégie sans affaiblir sa position personnelle. Dans un équilibre de Nash, les nœuds n'ont donc aucun intérêt à s'éloigner de l'équilibre de façon unilatérale. Les nœuds n'ont donc aucun intérêt à dévier du protocole si celui-ci est un équilibre de Nash. Il est à noter que la preuve que nous présentons est similaire et part des mêmes hypothèses que de nombreux travaux existants traitant du problème des nœuds rationnels : [8,9,10].

5.2.1 Modélisation des nœuds rationnels

Pour raisonner à propos du comportement des nœuds rationnels il est nécessaire de formaliser leur comportement. Dans le cas de *RAC*, le bénéfice qu'un nœud n peut tirer du système dépend des variables suivante :

- (A) La force de l'anonymat.
- (T) La capacité à envoyer des messages.
- (R) La capacité à recevoir les messages qui lui sont destinés.
- (F) Le nombre de messages à faire suivre.
- (C) Le nombre de messages à chiffrer.
- (D) Le nombre de messages à déchiffrer.

Les points A,T et R représentent les buts premiers d'un nœud lorsqu'il décide de participer au système : être capable de communiquer anonymement. Si un nœud n'a pas

ces objectifs il n'utilise pas *RAC*. Le point F correspond à une recherche d'efficacité d'un point de vu bande passante : un nœud rationnel peut souhaiter économiser de la bande passante pour la dédier à autre chose ou parce qu'il paye sa bande passante. Les points C et D correspondent à une recherche d'efficacité d'un point de vu temps de calcul : un nœud rationnel souhaite bénéficier des communications anonymes, mais s'il peut éviter de dépenser de l'énergie pour cela il n'hésitera pas à le faire. Le bénéfice d'un nœud n peut être défini à l'aide de l'équation suivante $B = \alpha A + \beta T + \gamma R - \delta F - \omega C - \phi D$, avec $\alpha \approx \beta \approx \gamma \gg \delta \approx \omega \approx \phi$. Cela signifie que les nœuds rationnels ne veulent pas économiser de la bande passante ou de temps processeur au dépend de leur anonymat ou de la transmission de leur messages.

Les nœuds rationnels sont aussi caractérisés par leur aversion aux risques, ils considèreront donc que les autres nœuds suivent le protocole et risquent de les expulser s'ils détectent la moindre anomalie. Ils considéreront aussi que l'adversaire cherche toujours à diminuer leur bénéfice. Pour finir, ils n'ont pas confiance en d'autres nœuds et ne forment donc pas de coalitions.

5.2.2 Preuve d'équilibre de Nash

Théorème 5.1. *Le protocole RAC assure un équilibre de Nash.*

Afin de prouver ce théorème, nous allons procéder de la même façon que les travaux existants dans ce domaine [8, 9, 10] : nous divisons le théorème en lemmes représentant les différentes étapes du protocole et, pour chacun de ces lemmes, nous expliquons pourquoi le bénéfice d'un nœud rationnel risque de baisser s'il dévie du protocole. La preuve du théorème découle directement de la preuve de chacun de ces lemmes.

Lemme 5.1. *Un nœud rationnel envoie toujours les messages à l'ensemble de ses successeurs dans le groupe ou la chaîne correspondant.*

Démonstration. Lorsqu'il s'agit de l'un de ses propres messages, un nœud rationnel veut maximiser les chances que ce message atteigne sa destination, il envoie donc le message à tous ses successeurs pour s'assurer que la diffusion va fonctionner correctement. Lorsqu'il s'agit d'un message qu'il doit faire suivre, un nœud rationnel pourrait être tenté de n'envoyer le message qu'à une sous partie de ses successeurs. Ceci lui permettrait d'économiser de la bande passante. Cependant, un nœud rationnel sait que la moitié de ses successeurs peuvent être contrôlés par l'adversaire. Il sait de plus que ces successeurs contrôlés par l'adversaire peuvent le mettre sur leur liste noire afin de le faire expulser et ainsi procéder à une attaque. Par conséquent, un nœud rationnel sait qu'il risque de se faire expulser et de perdre tout bénéfice si un seul de ses successeurs non contrôlé par l'adversaire met le nœud sur sa liste noire. Par conséquent, un nœud rationnel envoie toujours les messages à l'ensemble de ses successeurs dans le groupe ou dans la chaîne correspondant au message. □

Lemme 5.2. *Un nœud rationnel fait toujours suivre correctement les messages pour lesquels il est relais.*

Démonstration. Un nœud rationnel n pourrait être tenté de ne pas faire suivre un message pour lequel il est relais, dans le but d'économiser de la bande passante. Cependant, n sait que l'expéditeur d'un message dont il est relais, mettra n sur sa liste noire si

ce dernier ne fait pas suivre son message correctement. Sachant qu'il peut y avoir fG nœuds contrôlés par l'adversaire dans le groupe, et sachant que ces nœuds contrôlés par l'adversaire peuvent mettre n sur leur liste noire dans le seul but d'effectuer une attaque, n sait qu'il suffit qu'il soit sur la liste noire d'un nœud non contrôlé par l'adversaire pour pouvoir se faire expulser du système et ainsi perdre tout bénéfice. Par conséquent, n fait toujours suivre correctement les messages pour lesquels il est relais. □

Lemme 5.3. *Un nœud rationnel vérifie toujours que ses prédécesseurs envoient tous les messages une et une seule fois.*

Démonstration. Un prédécesseur n'envoyant pas un message peut être un adversaire essayant d'effectuer une attaque dite $N-1$ [32]. L'une des priorités d'un nœud rationnel étant de défendre son anonymat, il vérifiera toujours que l'ensemble de ses prédécesseurs lui envoient l'ensemble des messages qu'ils sont sensés envoyer. Un prédécesseur envoyant un même message deux fois peut être un adversaire essayant d'effectuer une attaque par répétition [66]. Un nœud rationnel vérifiera donc toujours que les messages ne sont pas envoyés deux fois, afin de défendre son anonymat. De plus, un nœud rationnel cherche à économiser ses ressources et notamment sa bande passante, il cherchera donc à éviter de recevoir ou faire suivre deux fois le même message. □

Lemme 5.4. *Un nœud rationnel envoie la liste des nœuds qu'il suspecte à chaque période de diffusion anonyme des listes noires.*

Démonstration. Comme montré dans Dissent v1 [4] le mécanisme de diffusion que nous utilisons permet de détecter toute non-participation ou mauvaise participation aux diffusions. Un nœud rationnel participera donc correctement au mécanisme afin de ne pas se faire expulser du système. De plus, les messages envoyés à l'aide de ce protocole ont une taille prédéfinie. Un nœud rationnel n'économisera donc pas de bande passante ou de temps processeur en envoyant de mauvaises informations. Par conséquent, il enverra sa liste noire. □

Lemme 5.5. *Un nœud rationnel diffuse les requêtes $JOINT$ qu'il reçoit de nœuds voulant joindre le système.*

Démonstration. Un nœud rationnel a tout intérêt à aider d'autres nœuds à joindre le système. En effet, cela lui assure que les nœuds contrôlés par l'adversaire ne sont pas les seuls à choisir qui peut entrer dans le système. De plus, si cette raison ne suffisait pas, il serait possible de mettre en place un mécanisme permettant de punir le nœud rationnel : si un nœud ne répond pas à la requête $JOINT$ d'un nœud entrant, ce dernier renvoi la requête, de plus en plus fréquemment, jusqu'à l'obtention d'une réponse. Un nœud rationnel cherchera à éviter cela car ce mécanisme consomme sa bande passante. Cela peut même aller jusqu'à le faire expulser du système, si sa bande passante est tellement utilisée qu'il n'est plus capable de communiquer correctement avec ses prédécesseurs et successeurs dans les chaînes et le groupe dont il fait partie. □

Lemme 5.6. *Un nœud rationnel envoie toujours des messages (ou du bruit) à l'ensemble de ses successeurs au débit requis par le protocole.*

Démonstration. Un nœud rationnel sait que s'il n'envoie pas de message ou de bruit au débit requis par le protocole ses successeurs le mettront sur leur liste noire. Il enverra donc des messages ou du bruit au débit requis par le protocole. Comme il a été prouvé précédemment qu'un nœud rationnel envoie toujours les messages à l'ensemble de ses successeurs, un nœud rationnel envoie toujours des messages ou du bruit à l'ensemble de ses successeurs au débit requis par le protocole □

Lemme 5.7. *Un nœud rationnel vérifie toujours que ses prédécesseurs envoient de nouveaux messages au débit requis par le protocole.*

Démonstration. Un nœud rationnel mettra sur sa liste noire un prédécesseur envoyant des messages à un débit supérieur à celui requis par le protocole, car cela lui coûte cher en bande passante (il doit recevoir et faire suivre ses messages). Il mettra aussi sur liste noire un prédécesseur envoyant de nouveaux messages à un débit inférieur au débit requis par le protocole, car cela peut réduire son anonymat (en faisant la différence entre son débit d'envoi et son débit de réception, il est possible d'en déduire le nombre de messages dont il est l'expéditeur) et être le signe d'un adversaire effectuant une attaque. □

5.3 Conclusion

Dans ce chapitre, nous avons prouvé que le protocole *RAC* assure l'anonymat des expéditeurs, l'anonymat des destinataires et l'anonymat des relations, aussi bien en présence d'un adversaire passif qu'en présence d'un adversaire actif. Nous avons ensuite prouvé que le protocole *RAC* était un équilibre de Nash et que les nœuds rationnels n'avait, par conséquent, pas intérêt à dévier du protocole.

6

Évaluation

Sommaire

Dans ce chapitre nous évaluons les performances de *RAC* tant au niveau du débit qu'au niveau de l'anonymat. Nous cherchons pour cela à répondre aux questions suivantes :

— Quel est le débit atteint par *RAC*, Dissent v1 et Dissent v2 ?
— Quelles sont les garanties d'anonymat fournies par *RAC*, Dissent v1 et Dissent v2 ?

Les évaluations se font à l'aide de simulations, pour le débit, et des formules présentées en chapitre 5, pour l'anonymat. La simulation nous a permis d'évaluer les performances du système pour un nombre de nœuds allant jusqu'à 100000 nœuds, ce qu'il était impossible de faire en déploiement réel. Nous commencerons ce chapitre en présentant l'environnement de simulation ainsi que les paramètres de configuration utilisés pour chacun des protocoles. Nous répondrons ensuite à chacune des deux questions précédentes.

6.1 Environnement de simulation

Les simulations ont été effectuées à l'aide d'Omnet++ [13]. Omnet++ est un outil de simulation écrit en C++ qui simule le comportement des nœuds et du réseau en découpant chaque étape en éléments distincts. Nous avons choisi d'utiliser ce simulateur car il a fait l'objet ou été utilisé dans de nombreuses publications [68, 69, 70, 71], nous apportant ainsi des garanties sur l'exactitude des résultats. Nous avons de plus comparé les résultats obtenus pour Dissent en simulation et en déploiement afin de s'assurer

que les résultats de simulation étaient correctes. A l'aide de ce simulateur nous avons simulé un réseau de nœuds connectés entre eux par un routeur. Chaque nœuds étant connecté au routeur par un lien de débit 1Gb/s. Nous avons utilisé cette configuration idéale car elle permet d'évaluer le débit maximum qu'il est possible d'atteindre avec chacun des protocoles.

6.2 Configuration des différents protocoles

Pour chaque évaluation nous avons testé deux configurations de *RAC* : une configuration sans groupes (tous les nœuds sont dans un même groupe) et une configuration avec des groupes de 1000 nœuds. Nous appelons respectivement ces configurations RAC-N et RAC-1000. Pour chacune des configurations, nous utilisons sept anneaux pour le protocole de diffusion : $R = 7$, et cinq relais pour les oignons : $L = 5$.

Dissent v1 ne possède pas de paramètres de configuration particuliers. Pour Dissent v2 nous utilisons le nombre optimal de serveurs. Comme présenté en chapitre 3, cela signifie que l'on choisit le nombre de serveurs de façon à minimiser à la fois l'utilisation du réseau entre les serveurs et les clients et l'utilisation du réseau entre les serveurs. c'est-à-dire que pour chaque valeur de N nous choisissons M de façon à minimiser $2 \times Bcast(\frac{N}{M}) + M \times Bcast(M)$.

6.3 Évaluation du débit

Afin d'évaluer le débit des différents protocoles, nous avons procédé comme suit. Dans un système composé de N nœuds, chaque nœud sélectionne au hasard un autre nœud et envoie des messages à ce nœud au débit maximum possible. Les messages ont une taille de 10kB. Le nombre de nœuds, N, est une variable de l'expérience. Pour chaque expérience, nous mesurons le débit moyen auquel chacun des N nœuds reçoit des messages. La figure 6.1 montre le débit mesuré pour RAC-N, RAC-1000, Dissent v1, Dissent v2 en fonction de N. A titre de comparaison, dans les mêmes conditions, oignon routing en pair-à-pair assure un débit de 200Mb/s.

La figure 6.1 montre tout d'abord que, lorsqu'il y a plus de 1000 nœuds dans le système, les deux configurations de *RAC* fournissent un meilleur débit que les deux configurations de Dissent. Par exemple, lorsqu'il y a 100000 nœuds dans le système le débit obtenu avec RAC-N est 15 fois supérieur au débit obtenu avec Dissent v2. Le débit obtenu avec RAC-1000 dans les mêmes conditions est 1300 fois supérieur à celui de Dissent v2. Pour finir, Dissent v1 est tellement lent qu'il n'est pas possible d'envoyer correctement un message avec 100000 nœuds.

Lorsque le nombre de nœuds est inférieur à 1000, le débit fourni par Dissent v2 est supérieur au débit fournit par RAC-N ou RAC-1000, mais il faut noter que le nombre de serveurs dans ces configurations varie de quatre (pour vingt machines) à quarante-cinq (pour mille machines). On est donc en droit de se demander s'il reste plausible d'avoir confiance en l'existence d'un serveur non contrôlé ou contrôlable par l'adversaire devant un si petit nombre de serveurs. On remarque aussi que, lorsque le nombre de nœuds est inférieur à 1000, le débit assuré par les deux configurations de *RAC* est le même. Cela s'explique par le fait que les deux configurations ne sont composées que d'un groupe lorsque le nombre de nœuds est inférieur à 1000.

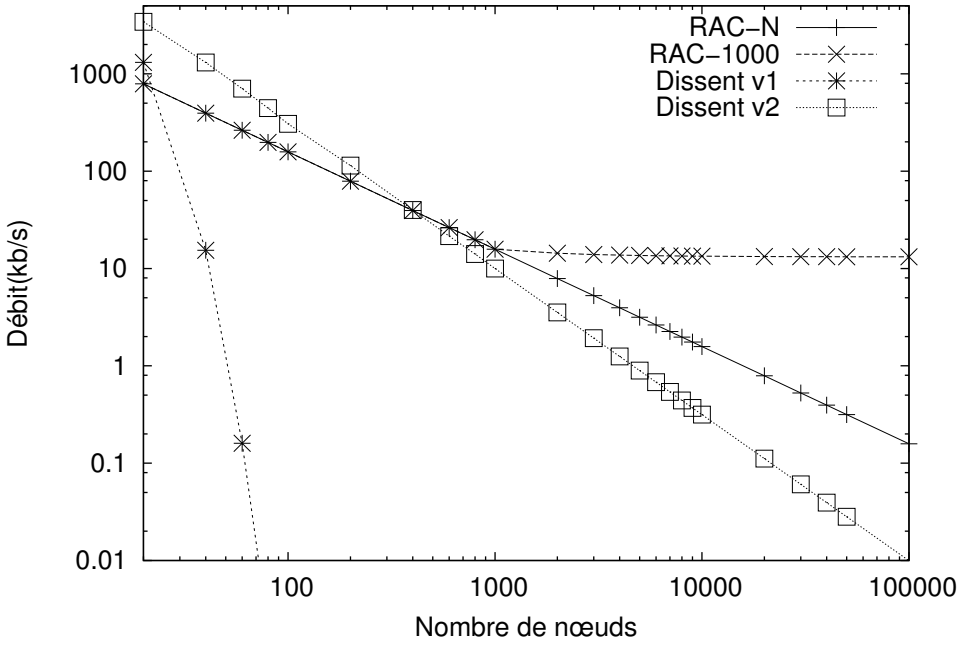


FIGURE 6.1 – Débit en fonction du nombre de nœuds dans le système, pour Dissent v1, Dissent v2, RAC-N et RAC-1000

La figure 6.1 montre finalement que RAC-1000 monte parfaitement en charge : dans un système comprenant plus de 1000 nœuds le débit de RAC-1000 ne dépend pas du nombre de nœuds dans le système. Ce résultat était attendu puisque la théorie montrait qu'ajouter des nœuds dans le système n'augmentait ni le nombre de diffusions nécessaires à l'envoi d'un message, ni le nombre de nœuds affectés par chaque diffusion.

En résumé, contrairement aux autres protocoles, le protocole *RAC* permet de dissocier le débit du nombre de nœuds et ainsi de monter en charge.

6.4 Garanties d'anonymat

Dans cette section nous comparons l'anonymat garanti par les deux configurations de *RAC* à l'anonymat garanti par Dissent v1, Dissent v2 et oignon routing. Ces comparaisons sont faites à l'aide des formules énoncées chapitre 5 dans le cas d'un opposant passif. Comme nous l'avons montré dans ce même chapitre, dans le cas de *RAC*, un opposant actif ne peut pas dévier du protocole de façon à réduire l'anonymat. L'anonymat fourni par *RAC* reste donc le même dans le cas d'un opposant actif et dans celui d'un opposant passif. Il en va de même pour Dissent qui ne permet pas aux nœuds de dévier du protocole. *Onion routing* sera par contre plus faible en cas d'opposant actif. Les résultats obtenus sont présentés dans le tableau 6.1.

La première ligne du tableau représente la taille du groupe auquel appartient un émetteur ou un destinataire. Plus ce groupe est grand, plus l'anonymat est important. En effet, plus cette valeur est petite plus l'opposant a d'information sur l'émetteur ou le destinataire d'un message. Par exemple, si la taille du groupe est 10, cela signifie

que la probabilité qu'un nœud soit l'émetteur ou le destinataire d'un message donné est $\frac{1}{10}$, alors qu'elle est de $\frac{1}{1000}$ lorsque la taille du groupe est de 1000. On observe dans le tableau que cette valeur est la plus élevée possible pour tous les protocoles sauf RAC-1000. La taille des groupes d'anonymat de ce dernier est égale à la taille des groupes : 1000. Nous pensons que cette taille est suffisante dans la plupart des cas, mais il faut noter que cette taille est une variable du système et qu'elle peut donc être augmentée si nécessaire.

Le tableau est ensuite divisé en trois sous-parties correspondant à la proportion (P) de nœuds du système contrôlés par l'adversaire. Nous étudions trois valeurs de P : 10%, 50% et 90%. Pour chacune de ses valeurs nous avons calculé la probabilité que l'opposant parvienne à casser l'anonymat d'un émetteur, d'un destinataire ou d'une relation. Les résultats montrent que ces probabilités sont égales à 0 pour Dissent v1 et Dissent v2. En effet, Dissent v1 et Dissent v2 s'appuient sur DC-Net. Il est donc prouvé que pour casser l'anonymat d'un nœud, l'adversaire doit contrôler tous les autres nœuds présents dans le système. Plus précisément, dans le cas de Dissent v2, l'opposant peut casser l'anonymat d'un client s'il contrôle tous les nœuds clients sauf un ou s'il contrôle tous les nœuds serveurs. Le deuxième cas est bien plus facile à atteindre car le nombre de serveurs est bien inférieur au nombre de clients, ici 447 serveurs pour 100000 clients. Le tableau montre aussi que, bien que parfois non nulles, les probabilités de casser l'anonymat sont très faibles pour les deux configurations de *RAC*. Ceci souligne le fait que *RAC* est un protocole robuste.

Le tableau montre aussi que, de façon contrintuitive, la probabilité que l'adversaire parvienne à casser l'anonymat d'un émetteur donné est plus faible dans RAC-1000 que dans RAC-N. Cela vient du fait qu'un nœud ne peut pas choisir le groupe qu'il va joindre dans RAC-1000. Par conséquent, l'opposant doit contrôler un grand nombre de nœuds dans le système pour espérer avoir assez de nœuds dans un même groupe pour casser l'anonymat d'un nœud de ce groupe.

Pour finir, on constate que, pour tous les types d'anonymat, RAC-1000 garantit un anonymat plus fort que celui d'*Onion routing*. Ceci pour deux raisons : premièrement, pour les raisons expliquées dans le paragraphe précédent, RAC-1000 garantit, grâce aux groupes, un anonymat des émetteurs plus important qu'*Onion routing* ; deuxièmement, l'anonymat des récepteurs de *RAC* est optimal (pour le casser l'opposant doit contrôler tous les nœuds d'un groupe sauf un) alors que l'anonymat d'*Onion routing* n'est pas optimal (il suffit à l'adversaire de contrôler tous les relais d'un oignon pour casser l'anonymat de son récepteur). La force de l'anonymat des communications découle de celle de l'anonymat des récepteurs.

Système contenant 100.000 nœuds			Dissent v1	Dissent v2	Onion	RAC-N	RAC-1000
Anonymat, l'émetteur/le destinataire est un parmi			100.000	100.000	100.000	100.000	1000
Probabilité de casser l'anonymat de type T d'un nœud donné en contrôlant P% des nœuds	% de nœuds adverses (P)	type d'anonymat (T)					
	90%	Émetteur	0	0	0.53	0.53	7.1×10^{-11}
		Destinataire	0	0	0.53	0	1.1×10^{-46}
		Relations	0	0	0.53	0	1.1×10^{-46}
	50%	Émetteur	0	0	1.5×10^{-2}	1.5×10^{-2}	1.8×10^{-16}
		Destinataire	0	0	1.5×10^{-2}	0	1.2×10^{-303}
		Relations	0	0	1.5×10^{-2}	0	1.2×10^{-303}
	10%	Émetteur	0	0	9.9×10^{-7}	9.9×10^{-7}	7.3×10^{-22}
		Destinataire	0	0	9.9×10^{-7}	0	5.8×10^{-1020}
		Relations	0	0	9.9×10^{-7}	0	5.8×10^{-1020}

TABLE 6.1 – Garanties d'anonymat des protocoles pour un système contenant 100000 nœuds

6.5 Conclusion

Nous avons montré que les performances de *RAC* en terme de débit étaient supérieure à celle de Dissent v1 comme v2 et que *RAC* permettait de monter en charge là où les débits de Dissent v1 et de Dissent v2 baissent avec le nombre de nœuds dans le système. Nous avons ensuite montré que si *RAC* assure un anonymat plus faible que celui assuré par Dissent v1 et Dissent v2, *RAC* assure un anonymat supérieur à celui assuré par *Onion routing* (le mécanisme d'anonymat le plus utilisé aujourd'hui).

Deuxième partie

Diffusion à ordre uniformément total.

7

Introduction

Sommaire

De plus en plus de tâches critiques sont déléguées aux ordinateurs. Il en résulte que la défaillance d'un ordinateur peut avoir des conséquences dramatiques. Malheureusement, la fiabilité des ordinateurs est loin d'être optimale. Cependant, les coûts réduits des machines ont permis d'assurer de la tolérance aux fautes grâce à la réplication de machines à états [14]. La réplication de machines à états consiste à maintenir plusieurs copies d'un même objet logiciel sur différentes machines de façon à ce que, si l'une des machines vient à être défaillante, il reste assez de copies pour que l'objet logiciel soit accessible. Les différentes machines sur lesquelles est copié l'objet sont appelées des réplicas. Afin de maintenir toutes les copies de l'objet logiciel dans le même état, il faut que les réplicas exécutent les requêtes qui leurs sont adressées dans le même ordre. Pour ce faire, chaque réplica diffuse les requêtes qu'il reçoit aux autres réplicas, à l'aide d'un protocole de diffusion à ordre uniformément total [72], et exécute les requêtes dans l'ordre dans lequel elles sont délivrées par le protocole de diffusion. Un protocole de diffusion à ordre uniformément total garantit les propriétés suivantes, pour l'ensemble des messages diffusés : (1) *Accord uniforme* : si une machine livre un message m alors tous les machines correctes finiront par livrer m ; (2) *Ordre total et uniforme fort* : si une machine livre le message m avant le message m', alors aucune machine ne livrera m' avant m.

Il est clair qu'il existe des applications pour lesquelles les garanties d'un mécanisme de diffusion à ordre uniformément total ne sont pas utiles. Ces applications peuvent par exemple se contenter de mécanismes de diffusion fiable ou même de mécanismes n'offrant pas de garanties de fiabilité. Cependant, nous verrons qu'il existe des protocoles de diffusion à ordre uniformément total, dont le nôtre, garantissant un débit égal au débit maximal possible. On peut donc dire que les garanties d'ordre uniformément total de notre protocole sont fournies gratuitement.

Dans la suite de ce chapitre, nous expliquons les notions de latence et débit. Nous présentons ensuite brièvement notre contribution. Enfin, nous mentionnons le plan suivi le long de cette partie.

7.1 Latence et débit

Il existe aujourd'hui de nombreux protocoles de diffusion à ordre uniformément total [16, 17, 73, 74, 75, 76, 77, 78, 79]. Une grande partie de ces protocoles a été conçue dans l'optique de garantir une latence faible. La latence correspond au temps nécessaire à la diffusion complète d'un unique message dans un système sans contention. Elle est mesurée en nombre de rondes dans le modèle de Lynch [80] : dans ce modèle chaque réplica peut envoyer un message au début de chaque ronde et reçoit les messages envoyés par les autres réplicas, à la fin de chaque ronde. D'un autre côté, quelques protocoles ont été conçus pour garantir un débit élevé. Le débit mesure le nombre de diffusions qu'un réplica peut effectuer par unité de temps. Le débit est important dans les environnements à forte charge tel que les systèmes de commerce électronique. En effet, lorsque la charge est importante, les messages peuvent passer un temps important dans des files d'attentes avant d'être diffusés. Les protocoles offrant un débit important permettent de réduire cette attente.

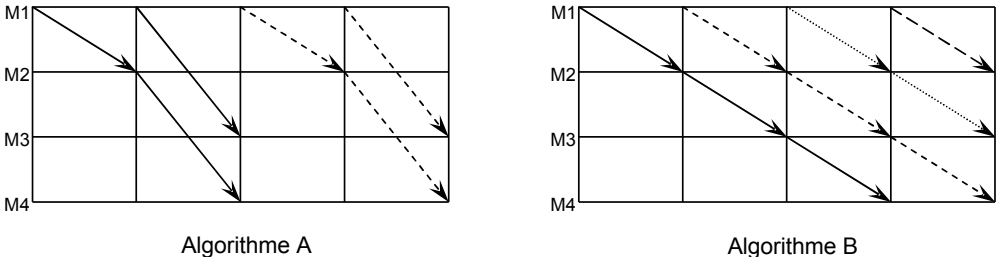


FIGURE 7.1 – Comparaison de deux algorithmes de diffusion, l'un privilégiant la latence (A), l'autre privilégiant le débit (B). La machine M_1 commence la diffusion. La latence est de 2 rondes pour A et de 3 pour B. Cependant, B a un débit plus élevé que A : l'algorithme B permet d'effectuer une diffusion par ronde alors que l'algorithme A ne permet que d'effectuer une diffusion toutes les 2 rondes

Les protocoles garantissant une latence faible ne parviennent souvent pas à assurer un débit élevé. Nous illustrons ce problème en figure 7.1 :

1. Dans l'algorithme A la machine M_1 commence par envoyer un message à M_2. Lors de la ronde suivante M_2 fait suivre le message à M_4 tandis que M_1 envoie le message à M_3.
2. Dans l'algorithme B la machine M_1 commence par envoyer un message à M_2. Lors de la ronde suivante M_2 fait suivre le message à M_3. Finalement M_3 fait suivre le message à M_4 lors d'une dernière ronde.

On voit que l'algorithme A a une latence de 2 (tous les réplicas ont reçu le message au bout de 2 rondes) alors que l'algorithme B a une latence de 3. Cependant l'algorithme

A ne permet au réplica p_1 de n'envoyer des messages que toutes les deux rondes alors que l'algorithme B permet au processus p_1 d'envoyer de nouveaux messages lors de chaque ronde. Par conséquent, bien que la latence de B soit supérieure à celle de A le débit de B est deux fois plus élevé que celui de A.

Afin de garantir une latence la plus faible possible, certains protocoles s'appuient sur la multi-diffusion IP. La multi-diffusion IP est un protocole réseau de bas niveau permettant à un émetteur d'envoyer un message à plusieurs destinataires simultanément. Ces protocoles ne parviennent pas à garantir un débit maximal pour la raison suivante : les messages envoyés par multi-diffusion IP sont abandonnés lorsque le réseau est congestionné. Pour éviter la congestion, les protocoles sont conçus de sorte qu'un seul réplica à la fois peut envoyer des messages. Hors, il a été montré par Guerraoui et al. [16] que ceci ne permet pas d'atteindre le débit maximal possible.

Il n'existe aujourd'hui qu'un protocole permettant d'atteindre un débit optimal : LCR [16]. Ce protocole organise les réplicas dans une topologie d'anneau : chaque réplica n'envoie des messages qu'à son successeur sur l'anneau. L'envoi se fait à l'aide d'un protocole fiable de communication point à point : TCP. Cette topologie permet d'atteindre un débit significativement supérieur au débit atteint par les protocoles à faible latence : le gain est par exemple de 25% dans un système contenant quatre réplicas. En contrepartie, la topologie en anneau entraine une augmentation linéaire de la latence avec le nombre de réplicas.

7.2 Contribution

Nous avons conçu *FastCast*, le premier protocole assurant à la fois un débit optimal et une latence faible. Afin de fournir une latence faible, *FastCast* utilise la multi-diffusion IP. Afin de fournir un débit optimal, *FastCast* permet à plusieurs réplicas d'envoyer des messages de façon simultanée. L'ordonnancement des messages est obtenu à l'aide d'un séquenceur fixe [15]. La nouveauté de *FastCast* réside dans un sous protocole exécuté par tous les réplicas et qui calcule de façon dynamique le débit auquel chaque réplica peut diffuser des messages.

Nous avons implémenté *FastCast* en C++ afin de comparer ses performances aux performances garanties par deux protocoles récents et constituant l'état de l'art : LCR [16] et Ring Paxos [17]. Le premier garantit un débit optimal tandis que le second tente de garantir à la fois un débit élevé et une latence faible. Notre évaluation sur un système contenant huit réplicas montre que *FastCast* garantit un débit optimal avec une latence faible. Plus précisément, *FastCast* garanti un débit jusqu'à 86% plus élevé que Ring Paxos et une latence jusqu'à 247% plus faible que LCR.

7.3 Plan

Le chapitre 8 présente les protocoles existants ainsi que l'environnement dans lequel nous allons les déployer. Nous décrivons ensuite le protocole *FastCast* en chapitre 9. Nous prouvons la correction de notre algorithme dans le chapitre 10. Puis nous évaluons ses performances au chapitre 11.

8

État de l'art

Sommaire

Il existe de nombreux protocoles de diffusion à ordre uniformément total. Avant de comparer ces protocoles, nous commencerons ce chapitre en définissant l'environnement dans lequel ces protocoles sont exécutés dans le cadre de cette étude. Nous verrons ensuite que les protocoles de diffusion à ordre uniformément total peuvent être classés en cinq classes de protocoles [15]. Nous présenterons chacune de ces classes afin d'en présenter les caractéristiques et les limites d'un point de vue performances. Une fois les différentes classes de protocoles présentées, nous présenterons, dans le détail, deux protocoles représentatifs de l'état de l'art : LCR [16] et Ring Paxos [17]. LCR est le seul protocole de diffusion à ordre uniformément total garantissant un débit optimal, tandis que, tel que montré dans [17], Ring Paxos est le seul protocole à "assurer un débit élevé tout en assurant une latence faible". Pour finir, nous conclurons sur la nécessité d'un nouveau protocole.

8.1 Environnement

Nous avons créé le protocole *FastCast* pour le déployer dans le cadre de grappes de machines identiques connectées entre elles par un réseau local. Nous partons de

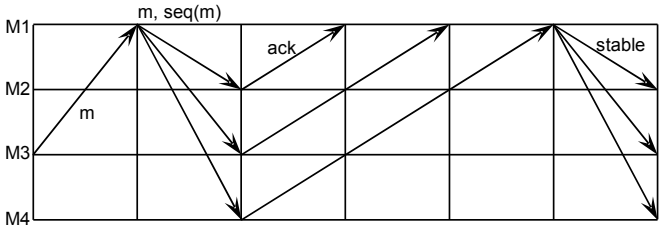
l'hypothèse que les machines ne peuvent dysfonctionner qu'en cessant de fonctionner (i.e. : les fautes Byzantines sortent du cadre de cette étude) et que chaque machine est équipée d'un détecteur de fautes parfait [81]. Un détecteur de fautes parfait donne la liste des machines fonctionnelles et garantit une précision forte (les machines fonctionnelles ne sont jamais suspectées d'être défaillantes). Afin d'implémenter un détecteur de fautes parfait, chaque machine crée une connexion TCP avec chaque autre machine et maintient cette connexion ouverte pendant l'intégralité de l'exécution du protocole (sauf si la machine cesse de fonctionner). Lorsqu'une connexion TCP est rompue, la machine essaye de la rétablir cinq fois. Si la machine ne parvient pas à rétablir la connexion, elle considère que l'autre machine a arrêté de fonctionner. Cette hypothèse est raisonnable, car dans une grappe, la latence du réseau connectant les machines entre elles est faible [82].

8.2 Catégories de protocoles existants

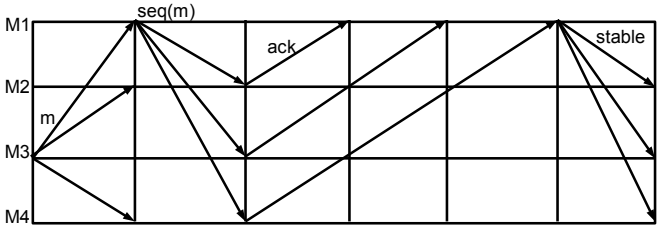
Les protocoles de diffusion à ordre uniformément total peuvent être classés en cinq classes [15] : les protocoles à séquenceur fixe, les protocoles à séquenceur mobile, les protocoles par privilèges, les protocoles par historique de communication et les protocoles par accord des destinataires. Dans cette section nous étudions les protocoles constituant l'état de l'art en nous appuyant sur ces cinq catégories. Nous n'étudions pas les protocoles faisant l'hypothèse d'une horloge synchronisée, car cette hypothèse n'est pas réaliste avec le matériel existant.

8.2.1 Protocoles par séquenceur fixe

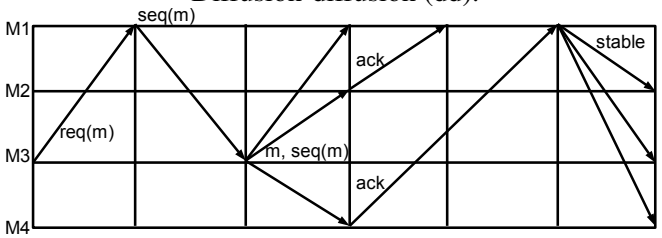
Les protocoles par séquenceur fixe [17, 73, 74, 75, 76, 78, 83] désignent une unique machine pour ordonner les messages. Une nouvelle machine n'est choisie pour ordonner les messages que si la précédente machine ayant ce rôle cesse de fonctionner. Baldoni et al. [84] ont montré que ces protocoles pouvaient être séparés en trois catégories selon leur schéma de communication : diffusion-diffusion (dd), envoi-diffusion (ed) et requête-diffusion (rd). Chacun de ces mécanismes est illustré par un exemple en figure 8.1. Les deux derniers types : ed et rd, sont souvent utilisés car ils assurent qu'une seule machine à la fois effectue des diffusions, ce qui garantit qu'il n'y aura pas de congestion sur le réseau. Le modèle ed est notamment le modèle utilisé par Ring Paxos [17], l'un des protocoles constituant l'état de l'art en matière de diffusion à ordre total uniforme. Cependant, le fait qu'il ne puisse y avoir qu'une machine à la fois qui envoie des messages réduit grandement le débit. Comme montré par Guerraoui et al. [16], il n'est pas possible d'atteindre le débit optimal permis par le réseau dans ces conditions. Même s'il n'existe pas aujourd'hui de protocole à séquenceur fixe assurant un débit optimal, nous verrons que *FastCast* permet d'atteindre le débit optimal en utilisant le modèle dd. D'un point de vue latence, il a été montré par Défago et al. [85] que la latence des protocoles à séquenceur fixe augmentait linéairement avec le nombre n de machines.



Envoie-diffusion (ed).



Diffusion-diffusion (dd).



Requête-diffusion (rd).

FIGURE 8.1 – Exemples de protocoles par séquenceur fixe.

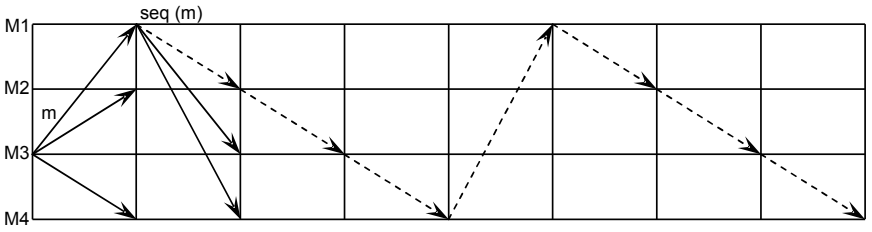


FIGURE 8.2 – Exemple de protocole par séquenceur mobile.

8.2.2 Protocoles par séquenceur mobile

Les protocoles par séquenceur mobile [86, 87, 88, 89], dont le principe est illustré sur la figure 8.2, fonctionnent sur le même principe que les protocoles à séquenceur fixe à l'exception du passage du séquenceur d'une machine à l'autre, même lorsque toutes les machines fonctionnent correctement. Le changement de séquenceur se fait par un passage de relais. L'idée est de répartir, sur toutes les machines, la charge qui incombe au séquenceur, afin d'éviter le goulot d'étranglement induit par le séquenceur fixe. Le fonctionnement est le suivant : lorsqu'une machine veut envoyer un message, elle le diffuse à l'ensemble des autres machines. À la réception d'un message les machines le mettent dans une file d'attente. Lorsqu'une machine devient séquenceur, elle assigne un ordre au premier message de la file et le diffuse. En plus d'informer toutes les machines de l'ordre du message, la diffusion joue aussi le rôle de passage de relais. Un séquenceur ne diffuse pas de nouvel ordre tant qu'il n'a pas reçu tous les messages et ordres diffusés avant qu'il ne devienne séquenceur. Le protocole garantit donc qu'un message peut être livré lorsque le relais a fait un tour complet. Ce dernier point entraine une latence plus élevée que les protocoles à séquenceur fixe [15]. En contrepartie, les protocoles à séquenceur mobile assurent un débit plus élevé que les protocoles à séquenceur fixe. Cependant, il n'existe actuellement pas de protocole à séquenceur mobile garantissant un débit optimal.

8.2.3 Protocoles par privilèges

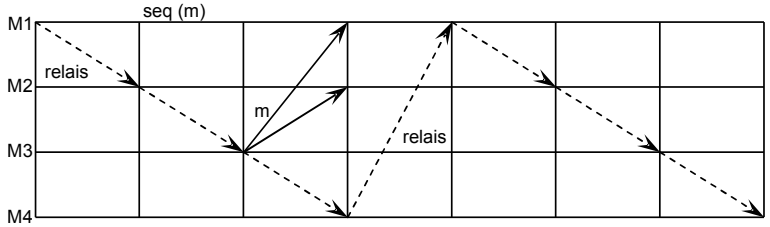


FIGURE 8.3 – Exemple de protocole par privilège.

Tout comme les protocoles à séquenceur mobile, les protocoles par priviléges [90,91, 92, 93, 94] s'appuient sur le passage d'un relais. Dans ces protocoles le relais n'indique pas qui est séquenceur, mais qui peut diffuser un message. Comme illustré en figure 8.3, seule la machine en possession du relais peut diffuser un message. Lorsqu'une machine reçoit le relais, elle vérifie qu'elle a reçu tous les messages précédemment envoyés avant d'envoyer son propre message. Un message peut donc être livré lorsque le relais a fait un tour complet. L'ordre des messages est donné par l'ordre des machines. Ce type de protocole ne permet qu'à une machine à la fois d'envoyer des messages. Il a donc été prouvé [16] qu'il était impossible d'atteindre le débit optimal avec ce type de protocole.

8.2.4 Protocoles par historique de communication

Les protocoles par historique de communication [16, 95, 96, 97, 98, 99] utilisent des horloges logiques pour ordonner les messages. Chaque machine peut envoyer

un message lorsqu'elle le souhaite. Les messages contiennent une horloge logique permettant aux machines recevant le message de déterminer les messages déjà reçu par l'envoyeur avant l'envoi du message. Ceci leur permet ensuite de déterminer quand le message doit être livré. De nombreux protocoles de cette catégorie [95, 96, 97, 98, 99] requièrent un nombre quadratique de messages échangés pour chaque message envoyé, ce qui entraine un mauvais débit. Cependant, LCR [16] utilise ce type d'ordonnancement et représente l'état de l'art en terme de débit : c'est le seul protocole assurant un débit optimal.

8.2.5 Protocoles par accord des destinataires

Dans ce type de protocoles, l'ordre de livraison est déterminé par un accord entre les destinataires du message. Il existe de nombreux protocoles de ce type [81, 100, 101, 102, 103]. Les mécanismes d'accord nécessitant un grand nombre d'échange de messages, ces protocoles ont tous des performances assez faibles. Leur principal intérêt est d'être créé pour des réseaux de grande taille plutôt que des grappes. Ils sont donc créés pour gérer des situations ne se présentant pas dans grappes.

8.3 Le protocole LCR

Comme mentionné précédemment, LCR est le seul protocole de l'état de l'art assurant un débit optimal. Comme prouvé par Guerraoui et al. [16], cela signifie que, dans un système contenant n machines, LCR assure (1) un débit égal à $\frac{n}{n-1}B$ (avec B le débit du réseau) lorsque les n machines diffusent des messages et (2) un débit égal à B lorsque moins de n machines diffusent des messages.

Afin d'assurer un débit optimal, LCR procède comme suit. Les n machines ($M_0 ... M_{n-1}$) sont organisées en anneau. Chaque machine a un prédécesseur et un successeur dans l'anneau : M_0 est précédée de M_{n-1} et suivie de M_1, M_1 est précédée de M_0 et suivie de M_2 et ainsi de suite. M_0 est appelée "première machine de l'anneau" et m_{n-1} est appelée "dernière machine de l'anneau". Lorsqu'une machine M_i souhaite envoyer un message m, elle l'envoie à son successeur qui le fait suivre à son successeur et ainsi de suite, jusqu'à ce que le message atteigne le prédécesseur de M_i, c'est-à-dire M_{i-1}. Afin d'assurer un ordre uniformément total, chaque machine ne doit livrer le message m que lorsqu'elle est sûre qu'elle a livré tous les messages devant être livrés avant m et que toutes les autes machines ont reçu m. Pour ce faire, lorsqu'une machine reçoit un message, elle place celui-ci en liste d'attente jusqu'à ce que ces deux conditions soient replies. Dans LCR, les machines vérifient que ces deux conditions sont remplies de la façon suivante : lorsque M_{i-1} reçoit le message m, elle envoit un message ACK à M_i, qui le fait suivre à M_{i+1} et ainsi de suite jusqu'à ce que le message atteigne *le prédécesseur du prédécesseur de* M_i, c'est-à-dire M_{i-2}. Lorsqu'une machine reçoit le message ACK, elle sait que : (1) m a été reçu par toutes les machines et (2) elle a reçu tous les messages devant être livrés avant m. Le premier point est évident : le message ACK est envoyé par M_{i-1}, cela signifie que M_{i-1} à reçu m, comme les messages sont transmis de successeur en successeur, cela signifie que m a fait le tour de l'anneau en passant par toutes les machines et a donc été reçu par toutes les machines. Afin d'expliquer le second point, il faut d'abord expliquer comment les

messages sont ordonnés dans LCR.

Dans LCR, l'ordre total est défini comme étant l'ordre dans lequel les messages sont reçus par la dernière machine de l'anneau, c'est-à-dire M_{n-1}. Pour illustrer cette relation d'ordre, considèrons deux messages m_i et m_j envoyés respectivement par les machines M_i et M_j, avec $i < j$ (ce qui signifie que M_i est avant M_j dans l'anneau). Sachant que les messages sont transmis de successeurs en successeurs à l'aide du protocole TCP, on sait que les messages ne peuvent pas se doubler lors de leur transmission. Il suffit donc de savoir si M_j avait reçu m_i avant d'envoyer m_j, pour savoir si M_{n-1} recevra m_i avant m_j et donc savoir si m_i doit être livré avant m_j. Afin de savoir si M_j avait reçu m_i avant d'envoyer m_j, les machines sont équipées d'une horloge vectorielle $C_{M_j} = (c_k)_{k=[0..n-1]}$. A chaque instant, la valeur $C_{p_j}[i]$ est égale au nombre de messages diffusés par M_i et reçus par M_j. Avant d'envoyer le message m_i, la machine M_i incrémente le nombre de messages qu'elle a diffusé (c'est-à-dire la valeur de $C_{p_i}[i]$) et inclue la valeur de son horloge vectorielle dans m_i. Lorsque M_j reçoit le message m_i, M_j met à jour son horloge vectorielle pour prendre en compte la reception de m_i : M_j incrémente la valeur de $C_{p_j}[i]$. Si, par la suite, M_j envoie un message m_j, ce message contiendra une représentation de l'horloge vectorielle de M_j témoignant de la réception du message m_i. Chaque machine peut obtenir l'ordonancement des messages m_i et m_j de la façon suivante. Si on note C_{m_i} la valeur de l'horloge vectorielle inclue dans m_i et C_{m_j} celle inclue dans m_j, le message m_i sera ordoné avant m_j, si est seulement si $C_{m_i}[i] \leq C_{m_j}[i]$ lorsque $i < j$, et $C_{m_i}[i] < C_{m_j}[i]$ lorsque $i = j$.

Lorsqu'une machine reçoit les messages ACK correspondant au message m_i, elle sait que : (1) m_i a été reçu par toutes les machines et (2) elle a reçu tous les messages devant être livrés avant m_i. Nous avons expliqué le point (1) précédement. Nous allons maintenant expliquer le point (2). Considérons un message m_j devant être livré avant m_i. Par définition de l'ordonancement dans LCR, nous savons que M_{n-1} recevra m_j avant m_i. Sachant que les messages ne peuvent pas se doubler pendant leur parcours sur l'anneau et sachant que m_i passera par M_{n-1} avant d'atteindre M_{i-1}, on sait que M_{i-1} recevra m_j avant m_i. On sait par conséquent que M_{i-1} recevra m_j avant d'envoyer le message ACK correspondant à m_i. Par conséquent, toutes les machines recevront m_j avant de recevoir le message ACK correspondant à m_i. Lorsque le message ACK correspondant à un message m est reçu, ce message m est marqué comme stable dans la liste d'attente et peut être livré dès que tous les messages le précédant dans la liste ont été livrés.

La topologie en anneau et le fait que les messages ne soient pas retransmis à leur émetteur (le message m_i est retransmis jusqu'à atteindre M_{i-1}) permet à LCR d'assurer un débit de $\frac{n}{n-1}B$. En effet, comme illustré sur la figure 8.4, si chaque machine diffuse des messages à un débit de $\frac{1}{n-1}B$, chaque machine envoie des messages sur le réseau à un débit de B : $\frac{1}{n-1}B$ pour diffuser ses propres messages, plus $\frac{n-2}{n-1}B$ pour faire suivre les messages des autres machines (n machines, moins la machine elle-même, moins le successeur de la machine). Chaque machine reçoit donc des messages de son prédécesseur à un débit de B et chaque machine reçoit ses propres messages (ne passant pas par le réseau) à un débit de $\frac{1}{n-1}B$, chaque machine livre donc des messages à un débit de $\frac{n}{n-1}B$. En contrepartie d'offrir un débit optimal, la topologie en anneau induit une latence élevée qui augmente linéairement avec le nombre de machines dans le système : chaque message doit être reçu puis retransmis par chaque

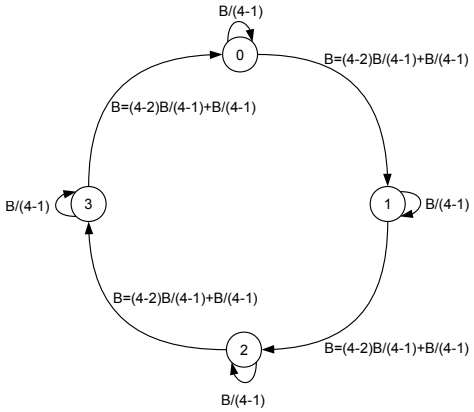


FIGURE 8.4 – Illustration des débit échangés dans LCR.

machine avant de pouvoir être livré. Nous avons évalué les performances de LCR dans un système contenant jusqu'à huit machines connectées entre elles par un switch 1Gb/s. Les figures 8.5 et 8.6 montrent respectivement le débit et la latence de LCR en fonction du nombre de machines. On peut voir que le débit est proche du débit optimal, mais que la latence devient vite importante.

8.4 Le protocole Ring Paxos

Après avoir contacté les problèmes de latence de LCR, Marandi et al. ont proposé un protocole garantissant à la fois un débit élevé et une latence faible : Ring Paxos. Pour obtenir ces performances, Ring Paxos implémente le protocole Paxos [104] en y intégrant quelques optimisations. Le protocole Paxos distingues quatre rôles pour les nœuds : les proposeurs, qui proposent des messages à diffuser ; les accepteurs, qui exécutent un mécanisme de consensus pour attribuer un ordre aux messages ; les apprenants, qui prennent en compte les messages des clients lorsque les accepteurs ont atteint un consensus ; et le coordinateur, qui est en fait un accepteur particulier permettant de garantir l'avancement du protocole. La principale optimisation qui différencie Ring Paxos des précédentes itérations faites sur Paxos est d'organiser les accepteurs dans une topologie d'anneau. Le coordinateur est aussi un accepteur et est placé à la fin de l'anneau. Organiser les accepteurs en anneau permet de réduire le nombre de messages que le coordinateur doit recevoir en les répartissant sur l'ensemble des accepteurs.

Le processus d'envoi de message de Ring Paxos est illustré figure 8.7. Lorsqu'une machine (un proposeur dans la figure) veut diffuser un message elle l'envoie au coordinateur. À la réception de ce message, le coordinateur le diffuse à l'aide de la multi-diffusion IP. Toutes les machines reçoivent le message et le mettent en liste d'attente. L'accepteur placé en premier dans l'anneau envoie une proposition d'ordre à son successeur dans l'anneau. Lorsque son successeur reçoit la proposition d'ordre, il vérifie qu'il a bien reçu le message correspondant à cette proposition d'ordre et que la proposition est correcte. Si ces deux conditions sont remplies, l'accepteur fait suivre la proposition d'ordre à l'accepteur suivant et ainsi de suite, jusqu'à ce que la proposition

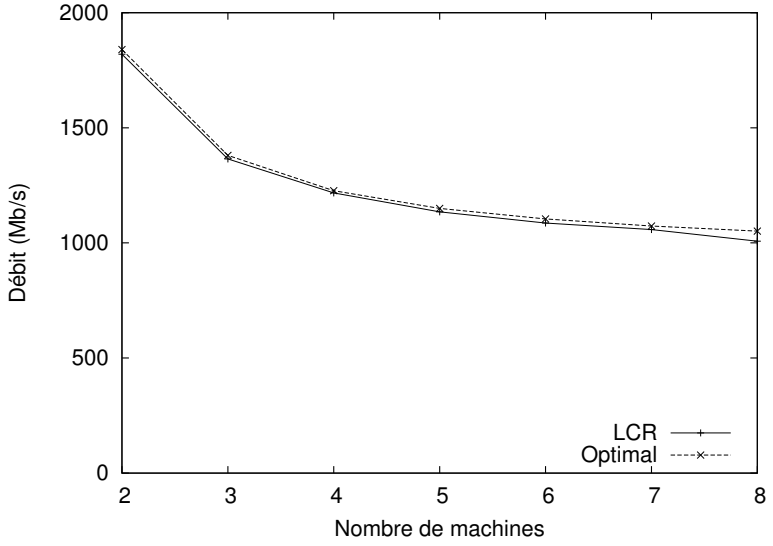


FIGURE 8.5 – Débit de LCR en fonction du nombre de machines dans le système.

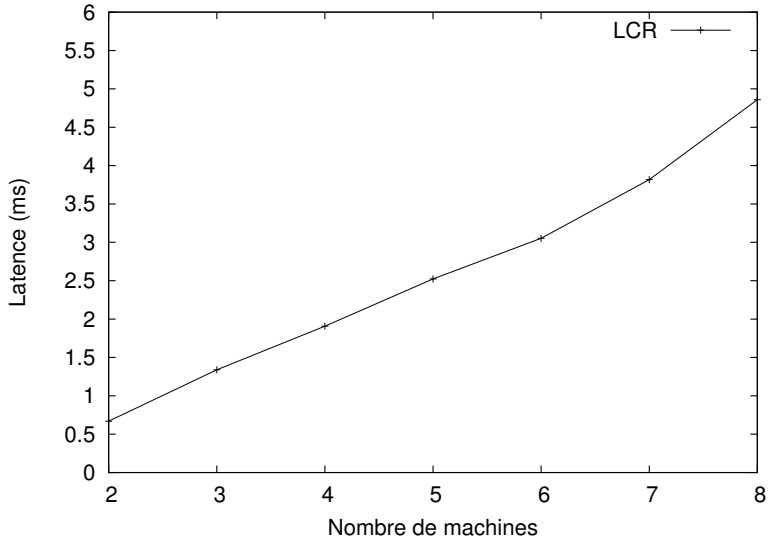


FIGURE 8.6 – Latence de LCR en fonction du nombre de machines dans le système.

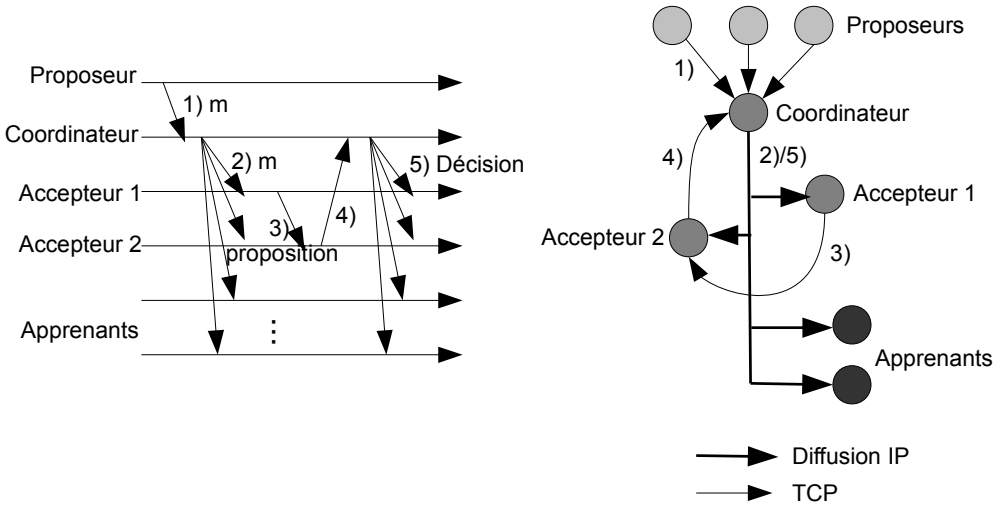


FIGURE 8.7 – Fonctionnement de Ring Paxos

d'ordre atteigne le dernier accepteur de l'anneau, c'est-à-dire le coordinateur. Lorsque le coordinateur reçoit la proposition d'ordre, il vérifie que celle-ci est correcte et, si elle est correcte, diffuse la décision à l'aide de la multi-diffusion IP. A la réception de la décision, toutes les machines connaissent l'ordonnancement du message et le livrent lorsqu'elles ont livré tous les messages précédents.

La topologie en forme d'anneau permet au coordinateur de ne recevoir que les propositions d'ordre vérifiées et envoyées par son prédécesseur. Dans les précédentes implémentations de Paxos, le coordinateur devait recevoir et vérifier les propositions d'ordre de chacun des accepteurs. Le fait de recevoir et traiter moins de messages de proposition d'ordre permet au coordinateur de consacrer la bande passante libérée à la réception de plus de messages à diffuser. Ceci permet à Ring Paxos de garantir un débit élevé. Cependant, tous les messages diffusés doivent être envoyés au coordinateur qui les diffuse ensuite, par multi-diffusion IP, à l'ensemble des machines. Comme montré par Guerraoui et al. [16], ceci ne permet pas d'assuer un débit optimal. Nous avons évalué les performances de Ring Paxos dans un système contenant jusqu'à huit machines connectées entre elles par un switch 1Gb/s. Comme le montre la figure 8.8, Ring Paxos garantit un débit proche du débit du réseau, mais loin du débit optimal. Concernant la latence, Ring Paxos n'oblige pas toutes les machines à faire partie de l'anneau des accepteurs, ce qui permet d'avoir une latence inférieure à celle de LCR. On peut par exemple considérer que la moitié des machines présentes dans le système sont correctes et qu'il sufit donc d'avoir $\frac{n}{2}$ accepteurs pour être sûr que le protocole fonctionne correctement. C'est l'expérience qui a été effectuée pour créer la figure 8.9. Comme on peut le constater, la latence augmente moins rapidement que celle de LCR, mais l'augmentation n'est néanmoins pas négligeable. On peut aussi constater que la latence augmente par palliers. Ceci vient du fait que, par exemple, lorsqu'il y a quatre ou cinq machines dans le système il faut trois machines dans l'anneau des accepteurs pour garantir le bon fonctionnement du protocole lorsque moins de la moitié des machines peuvent arrêter de fonctionner.

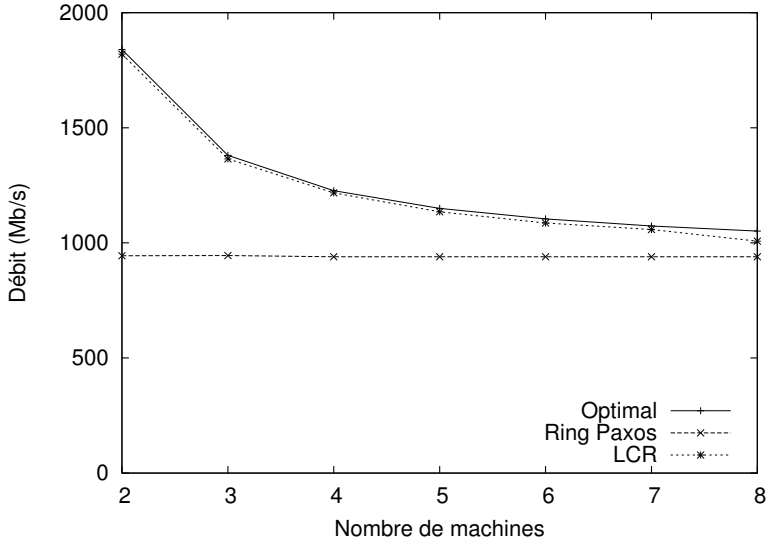


FIGURE 8.8 – Débit de Ring Paxos en fonction du nombre de machines dans le système.

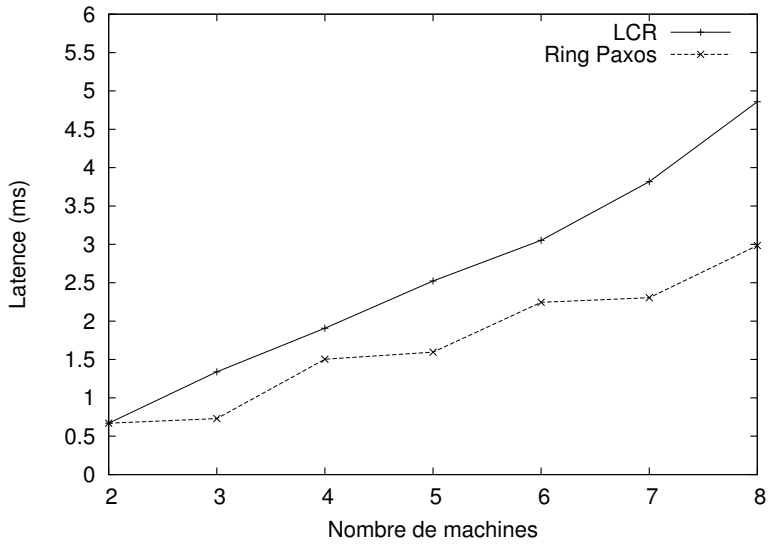


FIGURE 8.9 – Latence de Ring Paxos en fonction du nombre de machines dans le système.

En conclusion, le protocole Ring Paxos assure un débit élevé, mais sous-optimal, et garantit une latence faible, mais dont l'augmentation peut être rapide en fonction des contraintes sur le nombre d'accepteurs dans l'anneau.

8.5 Conclusion

Comme nous l'avons vu dans les sections précédentes, il existe un grand nombre de protocoles de diffusion à ordre uniformément total. Cependant, aucun de ces protocoles ne garantit à la fois un débit optimal et une latence faible. Il n'existe qu'un protocole garantissant un débit optimal : LCR. Mais la topologie en anneau permettant à LCR d'assurer un débit optimal est aussi la source d'une latence élevée et qui augmente rapidement lorsque le nombre de machines dans le système augmente. Ring Paxos a été proposé pour permettre un compromis entre une latence faible et un débit élevé. Mais on peut reprocher à ce protocole de ne pas garantir un débit optimal et d'avoir une latence qui augmente rapidement lorsque le nombre de machines dans le système augmente et lorsque les contraintes sur le nombre de machines dans l'anneau des accepteurs sont élevées.

Il faut donc un nouveau protocole de diffusion à ordre uniformément total, capable de garantir à la fois un débit optimal et une latence faible.

9

Le protocole *FastCast*

Sommaire

Dans ce chapitre nous décrivons le protocole *FastCast*. Nous commençons par donner une présentation générale du protocole avant de présenter les trois sous protocoles le composant.

9.1 Présentation générale

Notre but est de créer un protocole de diffusion à ordre uniformément total ayant un débit optimal tout en garantissant une latence faible. La meilleure option pour assurer une latence faible est d'utiliser de la multi-diffusion IP. En effet, la multi-diffusion IP permet à une machine d'envoyer un message à toutes les autres machines en un seul envoi. Ce choix est naturel et de nombreux protocoles de diffusion à ordre total s'appuient sur la multi-diffusion IP. Malheureusement, ce protocole d'envoi de paquets n'est pas fiable : des messages sont abandonnés lorsque le réseau est congestionné.

Afin de réduire le nombre de messages perdus, la plupart des protocoles existants s'appuient sur une technique simple : une seule machine à la fois peut envoyer des messages par multi-diffusion IP. De cette façon il est aisé d'éviter d'éventuelles congestions de réseau en contrôlant le débit d'envoi de la machine autorisée à faire de la multi-diffusion IP. Cependant, comme prouvé par Quema et al. [16], il est impossible d'atteindre un débit optimal en utilisant cette solution. Ce problème est illustré figure 9.1.

Dans cette illustration trois machines sont connectées entre elles par un switch 1Gb/s. Sur la partie gauche de la figure, une seule machine fait de la multi-diffusion IP. Le débit maximum auquel les machines peuvent diffuser dans ce cas est de 1Gb/s. Sur la partie droite de la figure, les trois machines font de la multi-diffusion IP en même temps. Chaque machine envoie à un débit de 500Mb/s. Dans cette configuration, le débit maximum auquel les machines livrent les messages est de 1.5Gb/S : chaque machine livre 500Mb/s qu'elle produit elle-même, plus 1Gb/s qui est envoyé par les autres machines. Ceci est possible grâce à deux faits : (i) les câbles réseau et les cartes réseau sont full-duplex (i.e. : les machines peuvent envoyer et recevoir des messages de façon simultanée) et (ii) le switch ne fait suivre les messages diffusés par multi-diffusion IP qu'aux machines autres que la source (i.e. : une machine ne reçoit pas ses propres messages via le réseau).

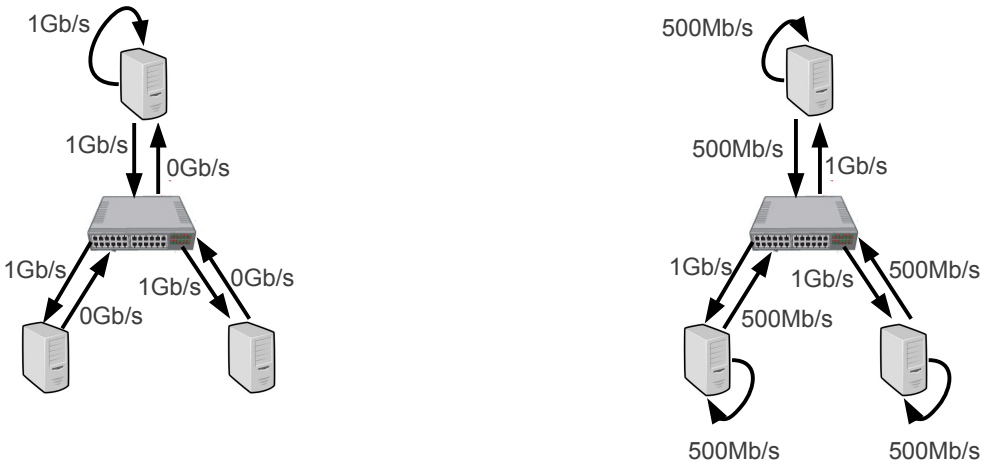


FIGURE 9.1 – Diffusion de messages (un émetteur sur la figure de gauche, plusieurs émetteurs sur la figure de droite) dans un système contenant trois machines.

Comme le but de *FastCast* est d'avoir un débit optimal tout en assurant une latence faible, le protocole permet à plusieurs machines de faire de la multi-diffusion IP simultanément. Le mécanisme d'ordonnancement de *FastCast* est un mécanisme à séquenceur fixe des plus basique que nous décrivons en sections 9.2 et 9.3. La nouveauté et le centre de *FastCast* est de proposer un sous-protocole permettant de synchroniser les débits d'envoi des différentes machines pour éviter les congestions et les pertes de messages qui en découlent. Ce sous-protocole, présenté en section 9.4, permet à toutes les machines de récupérer les demandes en bande passante des autres machines et d'adapter leur bande passante en fonction des demandes (à l'aide d'un algorithme dit de "max-min fairness" [105]). L'idée sous-tendant le protocole est simple et, comme montré chapitre 11, permet d'obtenir d'excellentes performances.

Dans la suite de ce chapitre, nous présentons les trois sous-protocoles qui constituent *FastCast*.

9.2 Le sous-protocole d'ordonnancement

FastCast propose deux primitives de communication à l'utilisateur : utoBroadcast et utoDeliver. Ces primitives assurent les quatre propriétés suivantes :

Validité : si une machine correcte M_i utoBroadcaste un message m, alors M_i finira par utoDelivrer m

Intégrité : pour tout message m, toute machine correcte M_i utoDelivrera m au plus une fois, et seulement si m a été utoBroadcasté précédemment par une machine M_i.

Accord uniforme : si une machine M_i utoDelivre un message m, alors toutes les machines correctes M_j finiront par utoDelivrer m.

Ordre total : pour tous message m et m', si une machine M_i utoDelivre m sans avoir utoDelivré m', alors il n'existera pas de machine M_j qui utoDelivrera m' avant m.

Le sous-protocole implémentant ces trois propriétés est présenté figure 9.2. C'est un protocole à séquenceur fixe classique [15]. Une machine est choisie pour *leader* et est chargée d'assigner et de diffuser les numéros d'ordonnancement. Il est important de noter que le leader n'est pas chargé de faire suivre les messages (appelé DATA message dans la figure 9.2). Chaque machine est chargée d'envoyer ses DATA message à toutes les autres machines (ligne 10). Afin d'assurer l'accord uniforme, chaque machine accuse réception de chaque message et de son numéro d'ordre (ligne 19 pour le leader et ligne 24 pour les autres machines). Chaque machine attend d'avoir reçu les accusés de réception de toutes les machines avant de livrer un message (ligne 30 et ligne 31). De cette façon, chaque machine est sûre que le message (ainsi que son numbéro d'ordre) a été reçu par toutes les autres machines et sera livré par toutes les machines correctes, même si la machine elle-même cesse de fonctionner. Notez que pour gérer la perte de messages, les machines utilisent un compte à rebours lorsqu'elles envoient un message (ligne 12). Si après un temps donné une machine n'a pas livré l'un de ses propres messages (i.e. : le message est toujours en liste d'attente (pendings), comme vérifié ligne 35), la machine renvoie le message (ligne 36).

9.3 Le sous-protocole d'adhésion

Afin de gérer l'arrivée et le départ de machines, *FastCast* est construit sur un protocole de communications de groupe [83] s'appuyant sur un détecteur de fautes parfait [81]. Les machines sont organisées en groupes qu'elles peuvent joindre ou quitter. Lorsqu'une machine joint ou quitte un groupe, cela déclenche un protocole de changement de "vue". Les machines arrêtant de fonctionner sont exclues de leur groupe grâce au détecteur de fautes parfait. Lors d'un changement de vue, les machines s'accordent sur une nouvelle vue : la vue courante v_r est remplacée par une nouvelle v_{r+1}.

La procédure $view\_change$ est détaillée figure 9.3. Il est à noter que lorsqu'une procédure de changement de vue est déclenchée, chaque machine commence par finir l'exécution de toute procédure de la figure 9.2 précédemment entamée. Une fois la procédure en cours finie, la machine commence la procédure de changement de vue.

Procédure exécutée par tous les p_i

```
 1: procedure initialize(initial_view)
 2:     pendings[] ← ∅
 3:     seqnos[] ← ∅
 4:     acks[][] ← ∅
 5:     snToDeliver ← 0
 6:     leader = p_0
 7:     sn ← 0

 8: procedure utoBroadcast(m)
 9:     id_m ← hash(p_i, m)
10:     Send ⟨DATA, id_m, m⟩ to all processes
11:     pendings[id_m] ← m
12:     SetTimeout ⟨id_m⟩

13: upon Receive ⟨DATA, id_m, m⟩ from p_j do
14:     if p_i = leader then
15:         if ∄ seqnos[id_m] then
16:             seqnos[id_m] ← sn
17:             sn ← sn + 1
18:             acks[id_m][p_i] = 1
19:         Send ⟨ACK, id_m, seqnos[id_m]⟩ to all processes
20:     pendings[id_m] ← m
21:     tryDeliver()

22: upon Receive ⟨ACK, id_m, sn_m⟩ from p_j do
23:     if p_j = leader and ∃ pendings[id_m] then
24:         Send ⟨ACK, id_m, sn_m⟩ to all processes
25:         acks[id_m][p_i] = 1
26:     seqnos[id_m] ← sn_m
27:     acks[id_m][p_j] = 1
28:     tryDeliver()

29: procedure tryDeliver()
30:     while ∃ id_m s.t. (seqnos[id_m] = snToDeliver and sum(acks[id_m]) = n) do
31:         utoDeliver(m)
32:         snToDeliver ← snToDeliver + 1
33:         pendings ← pendings − pendings[id_m]

34: upon Timeout⟨id_m⟩ do
35:     if ∃ pendings[id_m] then
36:         Send ⟨DATA, id_m, pendings[id_m]⟩ to all processes
37:         SetTimeout ⟨id_m⟩
```

FIGURE 9.2 – Pseudocode du sous-protocole d'ordonnancement.

Cette dernière fonctionne comme suit [1] : chaque machine envoie ses listes pending et seqnos à toutes les autres machines (ligne 2). À la réception de ces listes, chaque machine met à jour ses listes seqno et pending à l'aide des listes reçues (lignes 15 et 17). Après celà, la machine envoie un message ACK_RECOVER (ligne 18). Les machines attendent d'avoir reçu les messages ACK_RECOVER de toutes les autres machines (ligne 3) avant d'envoyer un message END_RECOVERY à toutes les autres machines (ligne 4). Lorsqu'une machine a reçu les messages END_RECOVERY de toutes les autres machines (lignes 5) elle peut livrer tous les messages pour lesquels elle a un ordre (lignes 19 à 24). De cette façon, toutes les machines de la nouvelle vue auront livré les mêmes messages dans le même ordre, à la fin du changement de vue. Chaque machine vide, ensuite, ses listes pendings, seqnos et acks (lignes 8 à 10). Pour finir, chaque machine prend la première machine de la nouvelle vue comme nouveau leader (ligne 11).

Procédure exécutée par tous les p_i

```
 1: upon view_change(new_view) do
 2:     Rsend ⟨RECOVER, p_i, pendings, seqnos⟩ to all p_j ∈ new_view
 3:     Wait until received ⟨ACK_RECOVER⟩ from all p_j ∈ new_view
 4:     Rsend ⟨END_RECOVERY⟩ to all p_j ∈ new_view
 5:     Wait until received ⟨END_RECOVERY⟩ from all p_j ∈ new_view
 6:     forceDeliver()
 7:     view ← new_view
 8:     pendings[] ← ∅
 9:     seqnos[] ← ∅
10:     acks[][] ← ∅
11:     leader = first process in view
12:     sn ← nextToDeliver

13: upon Rreceive ⟨RECOVER, pendings_{p_j}, seqnos_{p_j}⟩ from p_j do
14:     for each [id_m] ∈ pendings_{p_j} do
15:         pendings[id_m] ← pendings_{p_j}[id_m]
16:         if ∃ seqnos_{p_j}[id_m] then
17:             seqnos[id_m] ← seqnos_{p_j}[id_m]
18:     Rsend ⟨ACK_RECOVER⟩ to p_j

19: procedure forceDeliver()
20:     for each id_m ∈ seqnos[id_m], ordered by increasing sequence number do
21:         if ∃ pendings[id_m] and seqnos[id_m] ≥ snToDeliver then
22:             toDeliver(pendings[id_m])
23:             pendings ← pendings − pendings[id_m]
24:         snToDeliver ← seqnos[id_m] + 1
25:     for each id_m ∈ keys(pending[id_m]), ordered by increasing id_m do
26:         toDeliver(pendings[id_m])
27:         pendings ← pendings − pendings[id_m]
```

FIGURE 9.3 – Pseudocode du sous-protocole d'adhésion.

9.4 Le sous-protocole d'allocation de bande passante

Nous présentons, dans cette section, le mécanisme d'allocation de bande passante implémenté dans *FastCast*. Nous commençons par décrire les principes à la base de sa

1. notez que la fonction de changement de vue s'appuie sur deux primitives Rsend et Rreceive qui implémentent des canaux de communication fiable. Dans notre implémentation ces primitives sont réalisées à l'aide de TCP.

conception. Nous commentons ensuite son pseudocode. Pour finir, nous l'illustrons à l'aide d'un exemple.

9.4.1 Principes

Le but du sous-protocole d'allocation de bande passante est d'allouer de la bande passante à chaque machine de sorte que plusieurs machines puissent envoyer simultanément des messages par multi-diffusion IP sans qu'il y ait de perte de messages. Comme expliqué précédemment, il est indispensable que plusieurs machines puissent envoyer des messages simultanément pour que le débit soit optimal. En partant de l'hypothèse qu'à un instant donné toutes les machines connaissent la bande passante souhaitée par toutes les autres machines, il est facile d'allouer de la bande passante à chaque machine à l'aide d'un algorithme "max-min fair bandwidth allocation" [105]. Par exemple, considérons un système contenant trois machines connectées entre elles par un switch 1Gb/S et partons de l'hypothèse que chaque machine sait que la machine 1 souhaite avoir 700Mb/S, la machine 2 souhaite 600Mb/s et la machine 3 souhaite 300Mb/s. Chaque machine peut alors calculer de façon déterministe la répartition de bande passante suivante : 500Mb/s pour les machines 1 et 2 et 300Mb/s pour la machine 3. Il est en effet impossible d'attribuer plus de 500Mb/s aux machines 1 et 2. Sans quoi la machine 3 devrait recevoir des messages à un débit supérieur à 1GB/s, ce qui est impossible. En effet, le lien réseau entre la machine 3 et le switch est limité à 1Gb/s.

Il est possible de créer un protocole permettant aux machines d'échanger leurs souhaits en bande passante de façon à ce que toute machine sache, à tout moment, la bande passante souhaitée par toute autre machine. Cependant, un tel protocole serait couteux et nécessiterait la synchronisation de toutes les machines chaque fois qu'une machine change son souhait en bande passante. Cependant, il est possible d'allouer la bande passante de façon juste en s'appuyant sur des hypothèses plus faibles : il suffit que toutes les machines reçoivent les souhaits en bande passante dans le même ordre. Il est facile de remplir cette propriété en s'appuyant sur le protocole *FastCast* lui-même. Chaque fois qu'une machine veut changer son souhait en bande passante (i.e. : pour l'augmenter ou le diminuer), la machine envoie son nouveau souhait à toutes les autres machines à l'aide du protocole *FastCast*. De cette façon, toutes les machines recevront les messages de souhait de bande passante dans le même ordre.

Le dernier problème à résoudre est : quand une machine peut-elle modifier sa bande passante de manière effective ? Les machines se comportent différemment selon qu'elles souhaitent augmenter ou diminuer leur bande passante. Dans le cas d'une baisse de bande passante, la machine baisse effectivement la bande passante qu'elle utilise avant d'envoyer son souhait aux autres machines. De cette façon, lorsque les autres machines reçoivent le souhait, elles savent que la machine à déjà baissé sa bande passante. Elles peuvent donc calculer la nouvelle allocation de bande passante et augmenter leur propre consommation de bande passante, le cas échéant. Dans le cas d'une augmentation de bande passante, la machine n ne peut pas directement augmenter sa bande passante (sans cela l'augmentation pourrait provoquer une congestion du réseau). La machine commence donc par envoyer son souhait d'augmentation aux autres machines. A la réception du souhait, les autres machines calculent la nouvelle allocation de bande passante (en se basant sur le souhait de la machine) et diminuent leur propre bande passante, le cas échéant. Une fois la bande passante utilisée mise à jour, chaque machine

envoie un accusé de réception à n. Une fois qu'elle a reçu les accusés de réceptions de toutes les autres machines, n peut effectivement augmenter sa bande passante (en calculant localement la bande passante à laquelle elle a le droit).

9.4.2 Pseudocode

La figure 9.4 présente le pseudocode du sous-protocole d'allocation de bande passante. Chaque machine conserve en mémoire les souhaits en bande passante de toutes les autres machines dans la liste bwRequirements, ainsi que sa propre bande passante dans currentBW. Les variables ongoing_increase, deliver_req et acks sont utilisées lorsqu'une machine veut augmenter sa bande passante : ongoing_increase stocke la demande d'augmentation (avant d'être stockée dans bwRequirements lorsque toutes les machines auront accusé réception du souhait) ; le champ deliver_req indique si la machine elle-même a livré son propre message de souhait (si ce n'est pas le cas, la machine ne peut pas prendre en compte son propre souhait, même si elle a reçu tous les accusés de réceptions) ; pour finir, le champ ack permet de compter le nombre d'accusés de réceptions reçus pour la demande d'augmentation de bande passante en cours.

Avant d'entrer dans les détails du protocole, il faut noter que la fonction BW_allocation (lignes 34 à 47) est l'implémentation d'un algorithme classique de "max-min fair bandwidth allocation" [105]. Le seul point important de cette implémentation est la présence d'une variable, appelée availableBW, qui représente la capacité maximum des liens réseau. Cette capacité dépend de la taille moyenne des messages (il est bien connu que plus les messages sont large plus le débit maximum du protocole de communication est important [16, 17]). Dans notre implémentation nous utilisons des messages de taille moyenne 4kB et choisissons pour availableBW la capacité des liens réseau lorsqu'ils sont utilisés avec des messages de 4kB (cette capacité est proche de la valeur optimale). Pour s'assurer que c'est bien la capacité qu'auront les liens réseau pendant l'exécution du protocole, *FastCast* groupe les messages afin de garantir que les messages envoyés font au moins 4kB. Dans les cas où il n'y a pas de contention (la somme des souhaits est inférieure à la capacité du réseau) il est possible de ne pas grouper les messages, car le protocole n'a pas besoin d'atteindre un débit élevé.

Regardons maintenant le protocole d'allocation de bande passante. Une machine peut soit demander d'augmenter sa bande passante (en utilisant la fonction increase_BW, ligne 7), soit demander d'augmenter sa bande passante (en utilisant la fonction decrease_BW, ligne 26). Commençons par décrire ce qui se passe lorsqu'une machine souhaite augmenter sa bande passante. Pour augmenter sa bande passante, la machine appelle la fonction increase_BW. Au sein de cette fonction, la machine utoBroadcaste un message INCR à l'ensemble des autres machines (ligne 10). Lorsqu'elles livrent ce message, les autres machines mettent à jour leur liste bwRequirement (ligne 12), calculent leur nouvelle bande passante (ligne 13) en utilisant la fonction BW_allocation et envoient un accusé de réception (ACK) à la machine souhaitant augmenter sa bande passante (ligne 14). Lorsque la machine souhaitant augmenter sa bande passante a reçu les accusés de réception de toutes les autres machines et a livré son propre souhait (ligne 16), elle met à jour sa liste bwRequirements (ligne 21) et calcule sa nouvelle bande passante (ligne 22).

Voyons maintenant ce qui se passe lorsqu'une machine souhaite diminuer sa bande

passante. La machine utilise alors la fonction decrease_BW. Au sein de cette fonction, la machine met à jour sa liste bwRequirements (ligne 28) et calcule sa nouvelle bande passante (ligne 29). La machine utoBroadcaste ensuite un message DECR à l'ensemble des autres machines (ligne 30). Lorsque les autres machines livrent le message, elles mettent à jour leur liste bwRequirements (ligne 32) et calculent leur nouvelle bande passante (ligne 33) à l'aide de la fonction BW_allocation.

9.4.3 Illustration

Nous illustrons le fonctionnement du protocole à l'aide de trois exemples présentés dans les tableaux 9.1, 9.2 et 9.3. Nous considérons un système composé de trois machines connectées entre elles par un switch 1Gb/s. Dans chaque tableau nous décrivons les différentes étapes du protocole et montrons comment les différentes variables des trois machines sont mises à jour.

Les trois machines commencent avec une bande passante nulle (currentBW est égale à 0 dans le tableau 9.1, étape S1). Dans le tableau 9.1 nous décrivons ce qui se passe lorsque, en partant de cet état initial, la machine M_0 appelle la fonction increase_BW(800) et M_1 appelle la fonction increase_BW(300). À la fin des différentes étapes les machines atteignent un état (étape S8) dans lequel M_0 a la valeur 700Mb/s dans la variable currentBW et M_1 a la valeur 300Mb/s. En partant de cet état (aussi représenté dans le tableau 9.2, étape S9), le tableau 9.2 décrit les étapes suivant l'appel de increase_BW(600) par M_2. A la fin de ces étapes, les machines atteignent un état (étape S13) dans lequel M_0 et M_2 ont toutes deux la valeur 500Mb/s dans la variable currentBW et M_1 a la valeur 300Mb/s dans cette variable. Depuis cet état (aussi décrit dans le tableau 9.3, étape S14), le tableau 9.3 présente ce qu'il se passe lorsque M_2 appelle la fonction decreaseBW(500). Les machines atteignent un état (étape S16) dans lequel M_0 a la valeur 700Mb/s dans la variable currentBW, M_1 la valeur 300Mb/s et M_2 a la valeur 100Mb/s.

9.5 Conclusion

Dans ce chapitre nous avons présenté le protocole *FastCast*. Nous avons vu comment une utilisation intelligente de la multi-diffusion IP pouvait permettre d'assurer à la fois une latence faible et un débit optimal. Nous avons ensuite expliqué comment *FastCast* manageait le débit d'envoi des machines afin d'utiliser la multi-diffusion IP tout en évitant les pertes de messages. Pour finir, nous avons décrit en détails l'ensemble des sous-protocoles composant *FastCast* : le sous-protocole d'ordonnancement, le sous-protocole d'adhésion et le sous-protocole d'allocation de bande passante.

Dans les chapitres suivants nous commençons par prouver, dans un premier chapitre, que *FastCast* est bien un protocole de diffusion à ordre uniformément total et qu'il garantit bien un débit optimal, avant d'évaluer les performances du protocole, dans un second chapitre.

Procédure exécutée par tous les p_i

```
 1: procedure initialize(initial_view)
 2:     bwRequirements[] ← [0, ··· , 0]
 3:     currentBW ← 0
 4:     ongoing_increase ← 0
 5:     delivered_req ← false
 6:     acks ← 0

 7: procedure increase_BW(amount)
 8:     wait until ongoing_increase = 0
 9:     ongoing_increase ← amount
10:     utoBroadcast ⟨INCR, amount⟩ to all processes

11: upon utoDeliver ⟨INCR, amount⟩ from p_j ≠ p_i do
12:     bwRequirements[p_j] ← bwRequirements[p_j] + amount
13:     currentBW ← BW_allocation()
14:     Rsend ⟨ACK⟩ to p_j

15: upon utoDeliver ⟨INCR, amount⟩ from p_i do
16:     delivered_req ← true

17: upon Rreceive ⟨ACK⟩ from p_j do
18:     acks ← acks + 1
19:     if acks = N − 1 then
20:         wait until delivered_req = true
21:         bwRequirements[p_i] ← bwRequirements[p_i] + ongoing_increase
22:         currentBW ← BW_allocation()
23:         acks ← 0
24:         ongoing_increase ← 0
25:         delivered_req ← false

26: procedure decrease_BW(amount)
27:     wait until ongoing_increase = 0
28:     bwRequirements[p_i] ← bwRequirements[p_i] − amount
29:     currentBW ← BW_allocation()
30:     utoBroadcast ⟨DECR, amount⟩ to all processes

31: upon utoDeliver ⟨DECR, amount⟩ from p_j ≠ p_i do
32:     bwRequirements[p_j] ← bwRequirements[p_j] − amount
33:     currentBW ← BW_allocation()

34: function BW_allocation()
35:     nodes ← p_i and the (N-2) other biggest values in bwRequirements
36:     availableBW ← B
37:     do
38:     allocated = false
39:     for p_j in nodes do
40:         if bwRequirements[p_j] ≤ availableBW/size(nodes) then
41:             nodes ← nodes − p_j
42:             availableBW ← availableBW − bwRequirements[p_j]
43:             allocated = true
44:     while(nodes ≠ ∅ and allocated = true)
45:     if p_i ∈ nodes then
46:         return availableBW/size(nodes)
47:     return bwRequirements[p_i]
```

FIGURE 9.4 – Pseudocode du sous-protocole d'allocation de bande passante

step	*process*	$bwRequirements$	$currentBW$	$ongoing\_increase$	$acks$	$delivered\_req$	
S1	p_0	$[0,0,0]$	0	0	0	-	Initial state
	p_1	$[0,0,0]$	0	0	0	-	
	p_2	$[0,0,0]$	0	0	0	-	
S2	p_0	$[0,0,0]$	0	$\mathbf{800}$	0	-	p_0 calls increase_BW(800) p_1 calls increase_BW(300)
	p_1	$[0,0,0]$	0	$\mathbf{300}$	0	-	
	p_2	$[0,0,0]$	0	0	0	-	
S3	p_0	$[0,0,0]$	0	800	0	-	p_2 utoDelivers $\langle \text{INCR}, 800\rangle_{p_0}$ p_2 utoDelivers $\langle \text{INCR}, 300\rangle_{p_1}$
	p_1	$[0,0,0]$	0	300	0	-	
	p_2	$[\mathbf{800},\mathbf{300},0]$	0	0	0	-	
S4	p_0	$[0,0,0]$	0	800	$\mathbf{1}$	-	p_0 Rreceives $\langle \text{ACK}\rangle_{p_2}$ p_1 Rreceives $\langle \text{ACK}\rangle_{p_2}$
	p_1	$[0,0,0]$	0	300	$\mathbf{1}$	-	
	p_2	$[800,300,0]$	0	0	0	-	
S5	p_0	$[0,\mathbf{300},0]$	0	800	1	$\surd$	p_0 utoDelivers $\langle \text{INCR}, 800\rangle_{p_0}$ p_0 utoDelivers $\langle \text{INCR}, 300\rangle_{p_1}$
	p_1	$[0,0,0]$	0	300	1	-	
	p_2	$[800,300,0]$	0	0	0	-	
S6	p_0	$[0,300,0]$	0	800	1	$\surd$	p_1 Rreceives $\langle \text{ACK}\rangle_{p_0}$
	p_1	$[0,0,0]$	0	300	$\mathbf{2}$	-	
	p_2	$[800,300,0]$	0	0	0	-	
S7	p_0	$[0,300,0]$	0	800	1	$\surd$	p_1 utoDelivers $\langle \text{INCR}, 800\rangle_{p_0}$ p_1 utoDelivers $\langle \text{INCR}, 300\rangle_{p_1}$
	p_1	$[\mathbf{800},\mathbf{300},0]$	$\mathbf{300}$	$\mathbf{0}$	$\mathbf{0}$	-	
	p_2	$[800,300,0]$	0	0	0	-	
S8	p_0	$[\mathbf{800},300,0]$	$\mathbf{700}$	$\mathbf{0}$	$\mathbf{0}$	**-**	p_0 Rreceives $\langle \text{ACK}\rangle_{p_1}$
	p_1	$[800,300,0]$	300	0	0	-	
	p_2	$[800,300,0]$	0	0	0	-	

TABLE 9.1 – Un premier exemple du protocole d'allocation de bande passante.

step	*process*	*bwRequirements*	*currentBW*	*ongoing_increase*	*acks*	*delivered_req*	
S9	p_0	$[800, 300, 0]$	700	0	0	-	Initial state (equal to S8 in Table 9.1)
	p_1	$[800, 300, 0]$	300	0	0	-	
	p_2	$[800, 300, 0]$	0	0	0	-	
S10	p_0	$[800, 300, 0]$	700	0	0	-	p_2 calls increase_BW(600)
	p_1	$[800, 300, 0]$	300	0	0	-	
	p_2	$[800, 300, 0]$	0	**600**	0	-	
S11	p_0	$[800, 300, \mathbf{600}]$	**500**	0	0	-	p_0 utoDelivers $\langle \text{INCR}, 600 \rangle_{p_2}$
	p_1	$[800, 300, \mathbf{600}]$	**300**	0	0	-	p_1 utoDelivers $\langle \text{INCR}, 600 \rangle_{p_2}$
	p_2	$[800, 300, 0]$	0	600	0	$\surd$	p_2 utoDelivers $\langle \text{INCR}, 600 \rangle_{p_2}$
S12	p_0	$[800, 300, 600]$	500	0	0	-	p_2 Rreceives $\langle \text{ACK} \rangle_{p_0}$
	p_1	$[800, 300, 600]$	300	0	0	-	
	p_2	$[800, 300, 0]$	0	600	1	$\surd$	
S13	p_0	$[800, 300, 600]$	500	0	0	-	p_2 Rreceives $\langle \text{ACK} \rangle_{p_1}$
	p_1	$[800, 300, 600]$	300	0	0	-	
	p_2	$[800, 300, \mathbf{600}]$	**500**	**0**	**0**	-	

TABLE 9.2 – Un deuxième exemple du protocole d'allocation de bande passante.

step	*process*	*bwRequirements*	*currentBW*	*ongoing_increase*	*acks*	*delivered_req*	
S14	p_0	$[800, 300, 600]$	500	0	0	-	Initial state (equal to S13 in Table 9.2)
	p_1	$[800, 300, 600]$	300	0	0	-	
	p_2	$[800, 300, 600]$	500	0	0	-	
S15	p_0	$[800, 300, 600]$	500	0	0	-	p_2 calls decrease_BW(500)
	p_1	$[800, 300, 600]$	300	0	0	-	
	p_2	$[800, 300, \mathbf{100}]$	**100**	0	0	-	
S16	p_0	$[800, 300, \mathbf{100}]$	**700**	0	0	-	p_0 utoDelivers $\langle \text{DECR}, 500 \rangle_{p_2}$
	p_1	$[800, 300, \mathbf{100}]$	**300**	0	0	-	p_1 utoDelivers $\langle \text{DECR}, 500 \rangle_{p_2}$
	p_2	$[800, 300, 100]$	100	0	0	-	p_2 utoDelivers $\langle \text{DECR}, 500 \rangle_{p_2}$

TABLE 9.3 – Un troisième exemple du protocole d'allocation de bande passante.

10

Preuves

Sommaire

Dans ce chapitre, nous commençons par prouver que *FastCast* est un protocole de diffusion à ordre uniformément total. Nous prouvons ensuite que le mécanisme d'allocation de bande passante garantit que la bande passante allouée n'est jamais supérieure à la bande passante fournie par le réseau. Pour finir, nous prouvons que la bande passante allouée est égale à la bande passante maximum fournie par le réseau, lorsque les souhaits des différentes machines le permettent.

10.1 Preuve d'ordre uniformément total

Afin de prouver que *FastCast* est un protocole de diffusion à ordre uniformément total, nous prouvons que *FastCast* respecte les quatre propriétés mentionnées chapitre 9 section 9.2 : validité, intégrité, accord uniforme et ordre total.

Lemme 10.1. *Validité : si une machine correcte M_i utoBroadcaste un message m, alors M_i finira par utoDelivrer m.*

Démonstration. Soit M_i une machine correcte et m un message envoyé par M_i. Lorsque M_i envoie le message, elle l'ajoute à sa liste d'attente (ligne 11 figure 9.2). Si un changement de vue se produit, M_i sera dans la nouvelle vue, car M_i est correcte. Le sous-protocole d'adhésion garantit alors que M_i délivrera tous les messages présent dans sa liste d'attente (ligne 6 figure 9.3). Ainsi, M_i utoDelivre m s'il y a un changement de vue. Considérons maintenant le cas où il n'y a pas de changement de vue. Les messages restent dans la liste d'attente tant qu'ils ne sont pas utoDelivré (ligne 33

figure 9.2). De plus, ils sont régulièrement renvoyés par la machine émettrice (ligne 36 figure 9.2). Comme M_i est correcte elle renverra le message tant qu'il restera en liste d'attente. Les propriétés du réseau assurent qu'un message envoyé un nombre infini de fois sera reçu un nombre infini de fois. Toutes les machines recevront donc le message tant que celui-ci restera en liste d'attente. À chaque fois que le leader reçoit un message il envoie ou renvoie un accusé de réception donnant un ordre au message (ligne 19 figure 9.2). Toutes les machines recevront donc l'ordre du message, tant que celui-ci sera en liste d'attente de l'émetteur. A chaque fois qu'une machine reçoit un ordre pour un message qu'elle a déjà reçu elle envoie un accusé de réception (ligne 24 figure 9.2). M_i finira donc par recevoir tous les accusés de réception et finira donc par utoDelivrer m (ligne 31 figure 9.2). □

Lemme 10.2. *Intégrité : pour tout message m, toute machine correcte M_i utoDelivrera m au plus une fois, et seulement si m a été utoBroadcasté précédemment par une machine M_i.*

Démonstration. La seule panne pouvant affecter une machine est qu'elle cesse de fonctionner. Une machine ne peut donc pas exhiber un comportement arbitraire. Par conséquent, les machines n'utoDelivreront jamais un message n'ayant pas été préalablement utoBroadcasté. Le leader garde en mémoire l'ordre donné à chaque message et renvoie le même ordre lorsqu'il reçoit une nouvelle fois un message (figure 9.2 ligne 19). Si le leader est correct, chaque message aura donc un et un seul ordre et ne sera utoDelivré que lorsque $snToDeliver$ sera égal à cet ordre. Si le leader s'arrête et qu'il y a un changement de vue, le sous-protocole d'adhésion garantit que tous les ordres préalablement reçus par au moins une machine correcte seront envoyés à toutes les autres machines (figure 9.3 ligne 2) et que les listes d'attente seront vidées à la fin du changement de vue (figure 9.3, ligne 8). Un message ayant déjà été utoDelivré avant le début du changement de vue ne sera donc pas utoDelivré de nouveau, car il a un ordre inférieur à $snToDeliver$ (figure 9.3 ligne 22). Un message n'ayant pas été utoDelivré avant le début du changement de vue ne sera utoDelivré qu'une fois, car il est retiré de la liste d'attente après avoir été utoDélivré (figure 9.3 ligne 26). □

Lemme 10.3. *Accord uniforme : si une machine M_i utoDelivre un message m, alors toutes les machines correctes M_j finiront par utoDelivrer m.*

Démonstration. Une machine utoDelivre un message m dans deux cas : (1) lorsqu'elle a reçu les accusés de réception de toutes les autres machine et qu'elle a utoDelivré tous les messages précédant m ; (2) pendant un changement de vue. Dans le premier cas, le fait d'avoir reçu tous les accusés de réception garantit que toutes les machines ont reçu le message ainsi que l'ordre qui lui est attribué. Le fait d'avoir utoDelivré tous les messages précédant m assure que tous les accusés de réception de ces messages ont aussi été reçus. Par conséquent, une machine n'utoDelivre un message que lorsque toutes les machines ont reçu : (i) tous les messages précédant ce message, (ii) le message, et (iii) tous les ordres associés à ces messages. Toutes les machines utoDelivreront donc m lors de leur prochain $tryDeliver$ (figure 9.2 ligne 29) ou lors de leur prochain $forceDeliver$ (figure 9.3 ligne 19). Dans le second cas, le sous-protocole d'adhésion garantit que toutes les machines ont reçu toutes les listes d'attentes et les ordres des autres machines avant d'appeler $forceDeliver$ (figure 9.3 ligne 6). Le sous-protocole

garantit donc que toutes les machines ont la même liste d'attente lorsqu'elles appellent $forceDeliver$. Pendant l'exécution de $forceDeliver$, tous les messages contenus dans la liste d'attente et n'ayant pas été livrés auparavant sont livrés. Par conséquent, si une machine M_i utoDelivre un message m pendant cette phase, les autres machines ont deux possibilités : (i) elles ont déjà livré le message avant le changement de vue ou (2) elles vont livrer le message lorsqu'elles appelleront $forceDeliver$. □

Lemme 10.4. *Ordre total : pour tous messages m et m', si une machine M_i utoDelivre m sans avoir utoDelivré m', alors il n'existera pas de machine M_j qui utoDelivrera m' avant m.*

Démonstration. Lorsqu'il n'y a pas de changement de vue, si une machine utoDelivre un message m, cela signifie qu'elle a reçu tous les accusés de réception correspondant à ce message et à son ordre. Toutes les machines connaissent donc l'ordre associé à ce message. Le leader attribue des ordres croissants (figure 9.2 ligne 16) et les machines utoDelivrent les messages dans l'ordre qui leur est associé (figure 9.2 ligne 30 et figure 9.3 ligne 22). Par conséquent, si une machine délivre m avant d'avoir délivré m' c'est que soit m' a un ordre supérieur à m, auquel cas toutes les machines auront délivré m avant de délivrer m', soit m' n'a pas encore d'ordre et il en recevra un supérieur à m. Dans le cas d'un changement de vue, les machines commencent par s'échanger toutes les informations en leur possession (figure 9.3 ligne 2). Les machines sont donc toutes en possession des mêmes informations lorsqu'elles commencent à utoDelivrer les messages. Les machines commencent par utoDelivrer les messages pour lesquels elles ont un ordre (figure 9.3 ligne 22) avant d'utoDelivrer les messages pour lesquels elles n'ont pas d'ordre (figure 9.3 ligne 26). De plus, les machines utoDelivrent les messages ayant un ordre en suivant cet ordre(figure 9.3 ligne 22). Par conséquent, si une machine utoDelivre m avant m' et que m à un ordre, cela signifie soit que m' à un ordre supérieur à m soit que m' n'a pas d'ordre. Toutes les machines délivreront donc m avant m'. Si m n'a pas d'ordre et qu'une machine utoDelivre m avant m', alors m' n'a pas d'ordre non plus. Les machines utoDelivrent les messages n'ayant pas d'ordre en suivant l'ordre de leurs id_m (figure 9.3 ligne 26). Par conséquent, toutes les machines utoDelivreront donc m avant m'. □

10.2 Preuve du mécanisme d'allocation de bande passante

Dans cette section, nous commençons par prouver que le sous-protocole d'allocation de bande passante garantit que la bande passante totale utilisée n'est jamais supérieure à la bande passante B autorisée par le réseau. Nous prouvons ensuite que la bande passante totale utilisée est égale à B lorsque les souhaits des machines le permettent.

10.2.1 L'allocation n'excède pas la capacité du réseau

Afin de prouver que le sous-protocole d'allocation de bande passante garantit que la bande passante totale utilisée n'est jamais supérieure à la bande passante B autorisée par le réseau, nous partons d'un état initial où la bande passante totale utilisée est inférieure à B et nous montrons que, quelles que soit les opérations effectuées, la bande

passante totale utilisée reste inférieure à B. L'état initial considéré est plausible car à l'initialisation du protocole toutes les machines ont une bande passante de 0.

Une machine change sa bande passante dans quatre cas : (1) l'application appelle la fonction *decrease_BW*, (2) la machine utoDelivre un message INC, (3) la machine utoDelivre un message DECR, (4) la machine reçoit tous les accusés de réception d'un message INC qu'elle a envoyé et délivré. Les cas (1) et (2) entraînent tous deux une baisse de la bande passante utilisée. En effet, le cas (1) entraîne une baisse immédiate (figure 9.4 ligne 29) et le cas (2) peut soit entrainer une baisse de la bande passante utilisée (figure 9.4 ligne 13) soit ne causer aucun changement de bande passante.

Dans le cas (3), la machine m peut augmenter sa bande passante (figure 9.4 ligne 33). L'augmentation de bande passante est effectuée à partir de la liste de vœux. Les messages DECR étant totalement ordonnés, toutes les machines prenant en compte ce nouveau souhait auront une liste des vœux dans le même état. La seule exception est le cas ou m a envoyé un message INC et n'a pas encore reçu tous les accusés de réception correspondants. Dans ce cas la liste de vœux de m contient un souhait pour m inférieur à celui pris en compte par les autres machines. Ceci n'entraîne donc pas de risque de voir m augmenter sa bande passante en excès. Comme toutes les machines augmentent leur bande passante à partir d'une liste de vœux dans le même état, l'algorithme "max-min fair bandwidth allocation" [105] est appliqué aux mêmes valeurs par toutes les machines. Il n'y a donc pas de risque de voir une machine augmenter sa bande passante d'une valeur supérieure à celle correspondant à sa part de la bande passante libérée par la machine émettrice du message DECR. Cette bande passante ayant déjà été libérée par la machine émettrice du message, l'augmentation de bande passante de m n'entrainera pas une consommation totale de bande passante supérieure à celle consommée avant que la machine émettrice du message DECR ne baisse sa bande passante.

Dans le cas (4) la machine m augmente sa bande passante (figure 9.4 ligne 22). La machine ayant reçu tous les accusés de réception, il est garanti que toutes les machines ont calculé leur bande passante à l'aide d'une liste de vœux contenant le souhait de m. Les messages étant livré dans un ordre total, toutes les machines ont une liste de vœux contenant le souhait de m et tous les souhaits ordonnés avant m. Toutes les machines ont donc appliqué l'algorithme "max-min fair bandwidth allocation" [105] à une liste de vœux dans le même état que celle de m et ont déjà libéré la bande passante qui sera attribuée à m. L'augmentation de bande passante de m n'entrainera donc pas une consommation totale de bande passante supérieure à B.

Aucun des quatre cas de modification de la bande passante utilisée par une machine n'entrainant un dépassement de la bande passante totale consommée, nous pouvons conclure que le sous-protocole d'attribution de la bande passante garantit que la bande passante totale utilisée n'est jamais supérieure à B.

10.2.2 L'allocation atteint les limites du réseau

L'algorithme "max-min fair bandwidth allocation" [105] garantit qu'il allouera une bande passante totale égale à B s'il est appliqué à une liste de vœux le permettant. Les fonctions *utoBroadcast* et *Rsend* garantissent que tout message envoyé finira par être reçu. Nous pouvons donc dire que si toutes les machines ont émis un souhait tel que la liste des vœux permette d'obtenir B, toutes les machines finiront par recevoir tous les souhaits et avoir une liste de souhaits permettant d'obtenir B. Toutes les machines

appliqueront donc l'algorithme "max-min fair bandwidth allocation" [105] à une liste de souhaits dont la somme est supérieure à B. La bande passante totale utilisée sera donc égale à B.

10.3 Conclusion

Dans ce chapitre nous avons prouvé que *FastCast* fournissait bien les garantis d'un protocole de diffusion à ordre uniformément total. Nous avons ensuite prouvé que le mécanisme d'allocation de bande passant garantissait que la bande passante allouée n'est jamais supérieure à la bande passante fournie par le réseau. Nous avons prouvé, pour finir, que la bande passante allouée est égale à la bande passante maximum fournie par le réseau, lorsque les souhaits des différentes machines le permettent.

11

Évaluation

Sommaire

Dans ce chapitre nous évaluons les performances de *FastCast* et les comparons à deux protocoles représentant l'état de l'art : LCR [16] et Ring Paxos [17]. Ces deux protocoles garantissent une diffusion à ordre uniformément total. Nous avons choisi LCR car c'est le seul protocole existant garantissant un débit optimal [16]. Nous comparons également *FastCast* à Ring Paxos car, comme montré par Marandi et al. [17], ce protocole "garantit un débit très élevé tout en ayant une latence faible". Nos expériences n'évaluent que les phases de fonctionnement sans défaillances, car les défaillances sont très rares dans le genre d'environnement pour lequel le protocole a été développé.

Nous commençons ce chapitre par décrire l'environnement d'évaluation. Nous évaluons ensuite le sous-protocole d'allocation de bande passante de *FastCast*. Puis nous continuons avec deux sections comparant respectivement le débit, le temps de réponse et la latence de *FastCast*, LCR et Ring Paxos. Notre évaluation montre que *FastCast* est à la fois efficace en termes de débit et de latence. Plus précisément, *FastCast* a un débit aussi élevé que LCR et une latence plus faible que Ring Paxos.

11.1 Environnement d'évaluation

Les expériences ont été effectuées sur une grappe composée de huit machines octo-coeurs connectées entre elles par switch gigabit. Chaque machine est cadencée à 2.5GHz et est équipée de 16GB de RAM. Chaque machine tourne sur un noyau Linux

2.6.32. La bande passante réelle entre deux machines (mesurée avec Netperf [106]) est de 942Mb/s. Afin de garantir une évaluation équitable, nous avons implémenté *FastCast* et LCR en C++ en utilisant la même base de code que Ring Paxos. Pour finir, toutes les expériences présentées commencent par une période d'échauffement, suivie d'une période pendant laquelle les performances sont mesurées. Chaque période d'évaluation dure cinq minutes.

11.2 Evaluation du sous-protocole d'allocation de bande passante

Nous commençons par évaluer le sous-protocole d'allocation de bande passante de *FastCast*. Pour ce faire nous avons effectué l'expérience suivante. Nous avons déployé quatre machines qui envoient des messages de tailles variables allant de 1kB à 6kB. Les machines varient leur souhait en bande passante pendant l'expérience. Les machines commencent chacune avec un quart de la bande passante disponible. Apres 10s, la machine 0 diminue son souhait en bande passante. La machine 1 effectue la même opération après 20s. Après 30s, la machine 2 augmente son souhait en bande passante. Pour finir, la machine 0 augmente son souhait en bande passante à 40s. Les résultats de l'expérience sont présentés sur la figure 11.1. L'axe des abscisses présente le temps tandis que l'axe des ordonnées présente la bande passante allouée à chaque machine, la bande passante à laquelle chaque machine délivre des messages et la bande passante optimale possible. Nous pouvons observer que le débit auquel les machines délivrent les messages est très proche du débit optimal du système. Ceci prouve que le sous-protocole d'allocation de bande passante est efficace. Nous avons, de plus, utilisé cette expérience pour évaluer le temps nécessaire pour qu'une machine augmente sa bande passante, c'est-à-dire le temps passé entre le moment où une machine envoie son nouveau souhait aux autres machines et le moment où la machine peut effectivement augmenter sa bande passante. Après de nombreuses itération de cette expérience nous avons pu mesurer que le temps moyens nécessaire aux machines pour augmenter leur bande passante est de 3.8ms.

11.3 Evaluation du débit

Afin d'évaluer le débit des trois protocoles, nous avons effectué l'expérience suivante : nous déployons chaque protocole sur N machines qui envoient des messages au débit maximum permis par le protocole. La taille des messages est fixée à 10kB, ce qui permet d'atteindre le débit maximum possible avec chaque protocole. Chaque machine mesure le débit auquel elle livre les messages, le débit étant mesuré par le ratio de bits livrés sur le temps depuis la fin de l'échauffement. Le débit présenté figure 11.2 correspond à la moyenne des débits mesurés par chaque machines.

La figure 11.2 présente le débit atteint par *FastCast*, LCR et Ring Paxos lorsque le nombre de machines varie entre 2 et 8. A titre de comparaison, nous avons aussi tracé la courbe du débit optimal atteignable pour chaque configuration. Ce débit est égal à $\frac{N}{N-1}$ fois le débit maximum du réseau, qui est de 942Mb/s. Cette courbe nous permet de faire plusieurs constats. Premièrement, les débits de *FastCast* et de LCR sont proches du

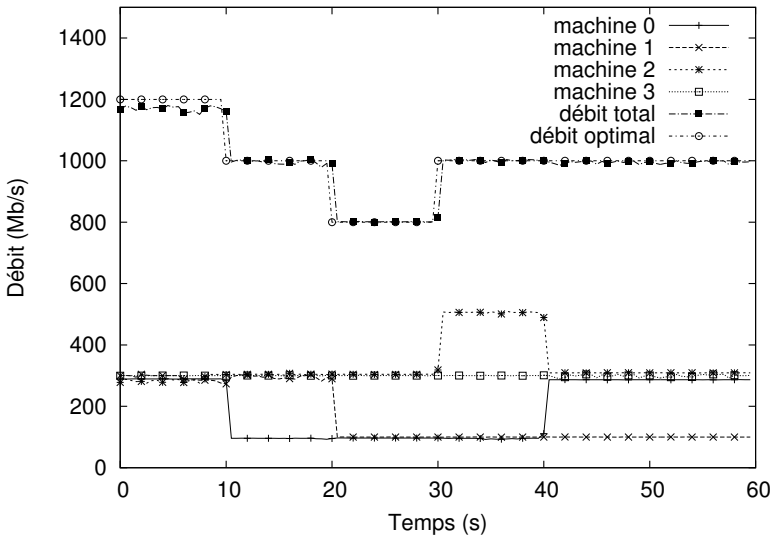


FIGURE 11.1 – Evaluation du sous-protocole d'allocation de bande passante.

débit optimal. Ceci confirme que, comme mentionné précédemment, le sous-protocole d'allocation de bande passante est efficace. Le débit de Ring Paxos est quasiment constant à 939Mb/s. Ceci était attendu, car dans Ring Paxos tous les messages doivent passer par le leader qui est le seul autorisé à faire de la multi-diffusion IP, ce qui limite le débit maximum du protocole. Ainsi, avec quatre machines *FastCast* et LCR sont 25% plus rapides que Ring Paxos. Dans un système contenant deux machines, *FastCast* et LCR sont 86% plus rapides que Ring Paxos.

11.4 Evaluation du temps de réponse

Dans cette section, nous évaluons le temps de réponse de *FastCast*, LCR et Ring Paxos dans un système contenant huit machines. Pour ce faire, nous varions le débit auquel les machines diffusent des messages. La taille des messages diffusés est de 10kB. Pendant la période d'évaluation, chaque émetteur mesure le temps écoulé entre l'envoi de chaque message m et la livraison de ce même message. Pour chaque protocole nous stoppons la mesure lorsque le débit atteint le débit maximum possible avec le protocole.

Les résultats de l'expérience sont montrés figure 11.3. L'axe des abscisses correspond au débit d'envoi total tandis que l'axe des ordonnées correspond au temps de réponse. On peut observer que *FastCast* garantit un temps de réponse bien inférieur à ceux de LCR et Ring Paxos. Plus précisément, *FastCast* garantit un temps de réponse jusqu'à 400% plus bas que celui de LCR et jusqu'à 246% plus bas que celui de Ring Paxos. Ceci est dû au fait que Ring Paxos et LCR s'appuient tout deux sur une topologie en forme d'anneau pour envoyer certains des messages nécessaires au protocole : les messages de DATA dans le cas de LCR et les messages d'ordonnancement dans le cas de Ring Paxos (il faut noter que contrairement à ce qui se fait dans LCR, Ring Paxos n'inclut pas toutes les machines dans l'anneau). Les messages doivent passer à travers l'ensemble des machines présentes dans les anneaux avant de pouvoir être livrés. Ceci entraine une augmentation du temps nécessaire au traitement des messages, par

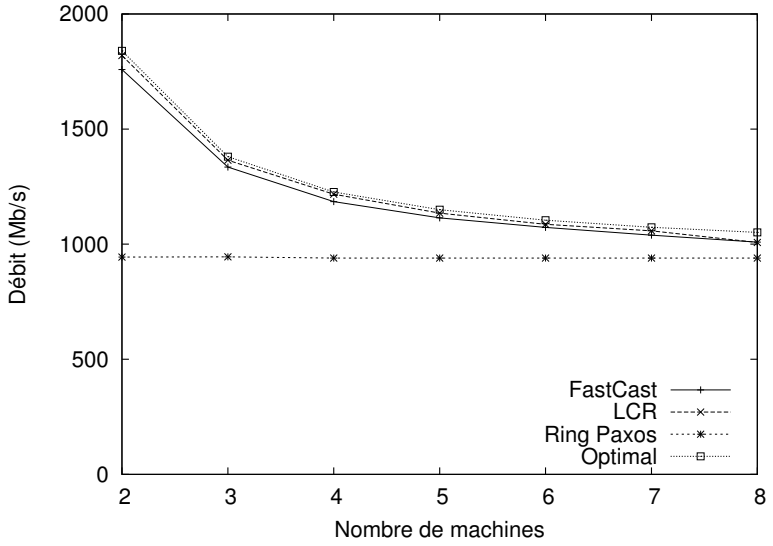


FIGURE 11.2 – Débit en fonction du nombre de machines, pour les protocoles *FastCast*, LCR et Ring Paxos.

rapport à un mécanisme comme *FastCast* dans lequel tous les messages sont envoyés par multi-diffusion IP sans suivre de motif de diffusion particulier.

11.5 Evaluation de la latence

Dans cette section nous évaluons la latence des protocoles *FastCast*, LCR et Ring Paxos. Pour ce faire, nous varions la taille du système de 2 à 8 machines. Rappelons que la latence correspond au temps nécessaire pour livrer un message lorsqu'il n'y a pas de contention. Afin de mesurer cette latence nous avons procédé comme suit : une machine diffuse des messages de 10 kB à un débit très bas (1 Mb/s). La machine envoyant les messages mesure le temps moyen entre le moment où elle diffuse chaque message et le moment où elle les livre.

Les résultats obtenus sont présentés dans la figure figure 11.4. L'axe des abscisses correspond au nombre de machines tandis que l'axe des ordonnées correspond à la latence. Nous pouvons observer que *FastCast* a une latence bien inférieure à celles de Ring Paxos et LCR. Plus précisément, *FastCast* a une latence jusqu'à 465% inférieure à celle de LCR et jusqu'à 247% plus basse que celle de Ring Paxos. De plus, nous constatons que la latence de *FastCast* est constante lorsque le nombre de machines augmente alors que celles de LCR et Ring Paxos augmentent avec le nombre de machines. L'augmentation de latence de Ring Paxos et LCR vient du fait que ces deux protocoles s'appuient sur une topologie en forme d'anneau. La latence de Ring Paxos n'augmente pas de façon linéaire, car dans Ring Paxos seule une majorité des machines est présente dans l'anneau. Ainsi, Ring Paxos a le même nombre de machines (trois) dans son anneau dans un système contenant quatre machines que dans un système contenant cinq machines. Dans les mêmes conditions l'anneau de LCR augmente linéairement avec le nombre de machines.

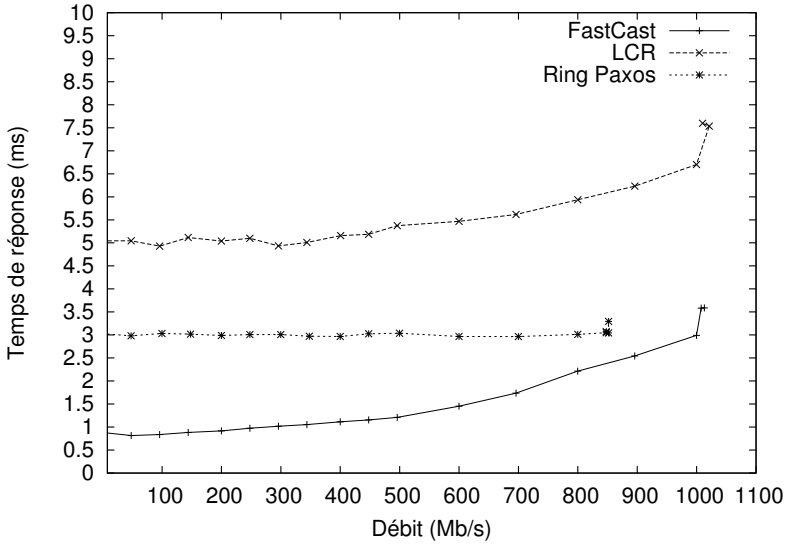


FIGURE 11.3 – Temps de réponse en fonction du débit d'envoi total pour les protocoles *FastCast*, LCR et Ring Paxos.

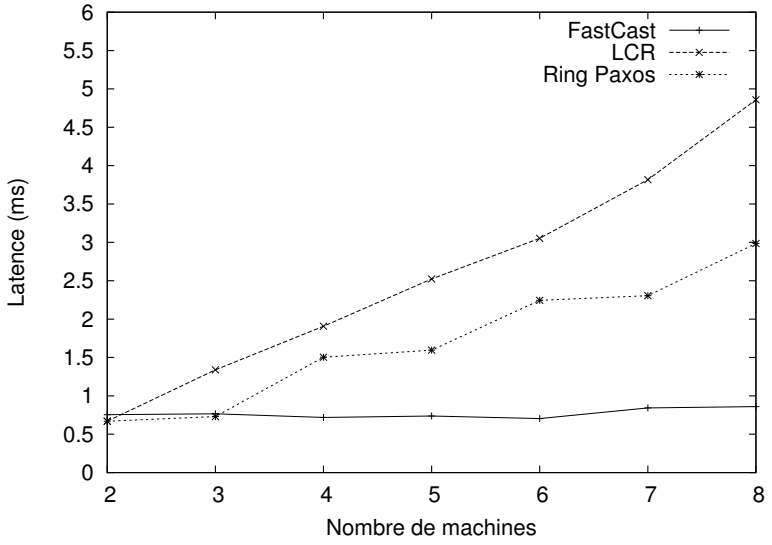


FIGURE 11.4 – Latence en fonction du nombre de machines, pour les protocoles *FastCast*, LCR et Ring Paxos.

11.6 Conclusion

Nous avons évalué *FastCast* sur une grappe de huit machines et avons comparé ses performances à celle de deux protocoles représentant l'état de l'art : LCR et Ring Paxos. L'évaluation montre que *FastCast* garantit un débit optimal et une latence très faible. Plus précisément, *FastCast* garantit un débit jusqu'à 86% supérieur à celui de Ring Paxos et une latence jusqu'à 247% inférieure à celle de LCR.

12

Conclusion

Sommaire

Pour conclure ce document nous commencerons par résumer les travaux effectués dans chacun des deux domaines étudiés, puis nous présentons de possibles ouvertures pour un futur travail.

12.1 Contributions

Durant cette thèse, nous nous sommes penchés sur les problèmes de transmission d'information dans les réseaux d'ordinateurs. Nous nous sommes plus particulièrement intéressés aux communications anonymes sur Internet et aux protocoles de diffusion à ordre uniformément total dans des grappes de machines. Nous rappelons nos contributions dans ces deux domaines dans les sections suivantes.

12.1.1 Les communications anonymes sur Internet

Les protocoles de communications anonymes permettent à des individus de communiquer sans qu'il soit possible d'identifier qui, au sein des utilisateurs du protocole, est source ou destination de chaque communication. L'idéal, pour que l'anonymat soit fort, est d'avoir un grand nombre d'utilisateurs et de fournir l'anonymat en s'appuyant exclusivement sur ces utilisateurs. En effet, plus le nombre d'utilisateurs est grand, plus chaque utilisateur est noyé dans la masse, plus il est compliqué de casser l'anonymat d'un utilisateur donné. De plus, s'appuyer exclusivement sur les utilisateurs permet

d'avoir une structure pair-à-pair plus compliquée à attaquer que si l'anonymat s'appuie sur un nombre restreint de serveurs. En effet, il faut alors attaquer tous les utilisateurs du système pour casser l'anonymat, alors qu'il suffit d'attaquer quelques serveurs dans le cas d'un protocole utilisant des serveurs. Cependant, quasiment toutes les solutions existantes aujourd'hui s'appuient sur l'existence de serveurs permettant d'assurer l'anonymat. Les protocoles de communications anonymes existants ne s'appuient pas sur le modèle pair-à-pair, car il n'existe pas de solution pour empêcher les nœuds d'agir rationnellement : les utilisateurs n'ont pas de véritable intérêt à aider les autres utilisateurs à être anonymes. Il en résulte que seuls les utilisateurs altruistes suivent le protocole et payent le prix résultant de la fourniture d'anonymat aux autres utilisateurs. Le nombre d'utilisateurs altruistes étant restreint, cette situation revient à fournir l'anonymat en s'appuyant sur un nombre restreint de serveurs.

Le seul protocole existant capable de prendre en charge les nœuds rationnels est Dissent v1 [4]. Dissent v1 [4], permet de garantir l'anonymat des communications de façon pair-à-pair en présence de nœuds rationnels. Cependant, il n'est pas capable de monter en charge et ne peut être utilisé avec plus de quarante utilisateurs. Ce chiffre étant trop faible pour garantir un anonymat raisonnable, nous avons proposé un nouveau protocole : *RAC*. *RAC* est le premier protocole de communications anonymes s'appuyant sur une structure pair-à-pair, fonctionnant en présence de nœuds rationnels et capable de monter en charge. Afin d'assurer un bon débit tout en étant capable de monter en charge, nous avons construit le mécanisme d'anonymat de *RAC* sur le modèle de celui d'*Onion routing* [1]. Nous avons ensuite construit un mécanisme de diffusion fonctionnant en présence de nœuds rationnels. Ce mécanisme, dont le design est inspiré de Fireflies [64], permet aux nœuds de surveiller anonymement le comportement des autres nœuds et de punir les nœuds rationnels. Pour finir, nous avons proposé de séparer les nœuds en groupes. De cette façon le coût de chaque diffusion n'est payé que par les nœuds du groupe de diffusion. La séparation des nœuds en groupes permet de faire un compromis entre niveau d'anonymat et performances : plus les groupes sont petits plus le débit est élevé ; plus les groupes sont gros plus l'anonymat est élevé (chaque utilisateur a un anonymat de "un parmi le nombre de membres de son groupe").

Nous avons comparé les performances de *RAC* à celles des protocoles de l'état de l'art en matière de communications anonymes en présence de nœuds rationnels : Dissent v1 [4], ainsi qu'à son évolution, Dissent v2 [5], qui a l'avantage de mieux monter en charge, mais s'appuie sur l'existence d'un nombre limité de serveurs pour assurer l'anonymat. Cette comparaison, faite par simulation à l'aide d'Omnet++ [13], a montré que *RAC* avait un meilleur débit que les deux versions de Dissent. Elle a aussi montré que *RAC* montait parfaitement en charge (sans perte de débit), alors que le débit des deux versions de Dissent décroit avec le nombre d'utilisateurs. Nous avons ensuite évalué mathématiquement les anonymats de *RAC*, Dissent v1 [4], Dissent v2 [5] et *Onion routing* [1]. Cette évaluation a montré que l'anonymat garanti par *RAC* est inférieur à celui garanti par Dissent, mais qu'il reste supérieur à celui garanti par *Onion routing*. Hors, *Onion routing* est le mécanisme d'anonymat le plus utilisé aujourd'hui, notamment dans Tor [2].

12.1.2 Les protocoles de diffusion à ordre uniformément total

Pour garantir de ne pas perdre d'informations ou de ne pas avoir de coupure de service les systèmes critiques tels que les banques, mais aussi les sites à forte audience s'appuient sur des mécanismes de réplication. Ces mécanismes consistent en l'exécution du même programme séquentiel sur plusieurs machines recevant les mêmes requêtes. De cette façon toutes les machines sont en permanence dans le même état. Si une machine vient à cesser de fonctionner, une autre machine peut prendre sa place sans qu'il n'y ait de perte d'informations ou d'inconsistance entre les réponses envoyées par la première machine et les réponses envoyées par la seconde machine. Pour que ces mécanismes fonctionnent il faut que toutes les machines traitent les requêtes dans le même ordre. Pour ce faire, les requêtes sont envoyées aux machines à l'aide d'un protocole de diffusion à ordre uniformément total. Les protocoles de diffusion à ordre uniformément total garantissent en effet que si un message est livré par une machine alors il sera livré par toutes les machines correctes et que si une machine livre un message m_1 avant un message m_2, alors toutes les machines livreront m_1 avant m_2.

Il existe de nombreux protocoles de diffusion à ordre uniformément total. Un grand nombre de ces protocoles sont optimisés pour garantir une bonne latence. C'est-à-dire qu'ils garantissent un intervalle de temps faible entre l'émission d'une requête et sa livraison par toutes les machines. Cependant, comme montré par Guerraoui et al [16], avoir une latence faible par message n'assure pas à ces protocoles la capacité de traiter un grand nombre de requêtes par seconde lorsque la charge du système est importante. Il n'existe qu'un protocole garantissant un débit optimal, LCR [16]. Ce protocole garantit que lorsque le système est utilisé à son maximum, le nombre de requêtes livrées par seconde est égal au nombre maximum qu'il est possible d'atteindre étant donné les limites du réseau. En contrepartie la latence de LCR est élevée. Il n'existe aujourd'hui aucun protocole permettant d'avoir à la fois une latence faible est un débit optimal.

Afin de pallier ce manque, nous avons proposé *FastCast*, le premier protocole de diffusion à ordre uniformément total garantissant à la fois une latence faible et un débit optimal. Afin de garantir une latence faible, nous nous sommes appuyés sur le mécanisme de multi-diffusion IP. Cette solution est utilisée par de nombreux protocoles de diffusion à ordre uniformément total. Nous nous différencions de ces protocoles dans la façon dont nous pallions les pertes de messages que connait ce mécanisme lorsque le réseau est surchargé. Les protocoles existants évitent la surcharge du réseau en limitant le nombre de machines autorisées à faire de la multi-diffusion IP. Or, il a été montré par Guerraoui et al [16] qu'il était impossible d'obtenir un débit optimal si toutes les machines ne pouvaient pas diffuser de messages simultanément. Nous avons donc proposé un protocole d'allocation de bande passante permettant à toutes les machines d'effectuer de la multi-diffusion IP tout en assurant que le réseau ne soit jamais saturé.

Nous avons comparé *FastCast* à deux protocoles représentant l'état de l'art en matière de protocoles de diffusion à ordre uniformément total : LCR [16] et Ring Paxos [17]. LCR est le seul protocole à garantir un débit optimal, tandis que Ring Paxos est le protocole fournissant le meilleur débit parmi les protocoles garantissant une latence faible. Pour effectuer cette comparaison, nous avons implémenté *FastCast* et LCR en C++ en utilisant la même base de code que Ring Paxos. Nous avons ensuite exécuté ces trois implémentations sur huit machines connectées entre elles par un switch

gigabit. La comparaison a montré que le débit garanti par *FastCast* est optimal tandis que sa latence est plus faible que celle de Ring Paxos. *FastCast* garantit ainsi un débit jusqu'à 86% plus élevé que Ring Paxos et une latence jusqu'à 247% plus faible que LCR.

12.2 Ouvertures

Pour chacun des deux travaux effectués il est possible de pousser plus avant les études et d'étendre la portée du travail effectué.

12.2.1 Les communications anonymes sur Internet

Le protocole *RAC* empêche les nœuds rationnels de dévier du protocole et garantit que les nœuds contrôlés par l'adversaire ne pourront pas casser l'anonymat en déviant du protocole. Cependant, il est possible qu'il existe des comportements byzantins n'étant pas détectés et stoppés par le protocole. Afin de rendre *RAC* plus robuste, il est nécessaire d'identifier l'ensemble de ces comportements et de proposer des mécanismes assurant l'expulsion des nœuds byzantins. Par ailleurs, d'un point de vue plus pratique, il serait intéressant d'effectuer une vraie implémentation de *RAC* afin de l'évaluer non plus en simulations mais en conditions réelles.

Pour finir, nous pensons qu'au vue des révélations récentes sur PRISM [1], les protocoles de communications anonymes vont être de plus en plus utilisés et qu'ils devront se tourner vers une structure pair-à-pair pour assurer un anonymat fort tout en étant capable de supporter l'afflux de nouveaux utilisateurs. Il nous semble donc important d'identifier les possibles optimisations de *RAC* permettant d'assurer un meilleur débit.

12.2.2 Les protocoles de diffusion à ordre uniformément total

FastCast a été créé pour assurer un débit optimal et une latence faible dans une grappe de machines connectées entre elles par un switch. Afin de rendre ce protocole plus polyvalent, il est nécessaire de rendre le protocole adaptable à d'autres topologies réseau, comportant par exemple plusieurs switch et de possibles goulots d'étranglement. Une première étape serait d'améliorer le mécanisme d'allocation de bande passante de sorte qu'il base sa répartition sur une topologie fournie par l'utilisateur. Une seconde étape consisterait à rendre automatique la détection de cette topologie et de permettre au mécanisme d'allocation de bande passante de s'adapter à d'éventuels changements à chaud de topologie.

Une seconde piste d'amélioration pour *FastCast* est la prise en compte du trafic présent en arrière-plan sur le réseau. En effet, *FastCast* utilise l'ensemble des capacités réseau et ne permet pas de prendre en compte le trafic en provenance d'autres applications ou d'autres machines utilisant le même réseau. Une solution qui nous parait être la plus évidente et la plus simple consisterait à appliquer le mécanisme d'allocation de bande passante à l'ensemble du trafic présent sur le réseau. Il nous semble nécessaire d'évaluer la faisabilité et le coût de cette solution, ainsi que d'explorer d'autres pistes afin de trouver la solution la plus efficace et la moins coûteuse en performances.

1. Prism : http ://fr.wikipedia.org/wiki/PRISM_%28programme_de_surveillance%29

Bibliographie

[1] D. Goldschlag, M. Reed, and P. Syverson, “Onion routing,” *Commun. ACM*, vol. 42, no. 2, pp. 39–41, Feb. 1999. [Online]. Available : http://doi.acm.org/10.1145/293411.293443

[2] R. Dingledine, N. Mathewson, and P. F. Syverson, “Tor : The second-generation onion router,” in *USENIX Security Symposium*, 2004, pp. 303–320. [Online]. Available : http://www.usenix.org/publications/library/proceedings/sec04/tech/dingledine.html

[3] D. Chaum, “The dining cryptographers problem : Unconditional sender and recipient untraceability,” *Journal of Cryptology*, vol. 1, no. 1, pp. 65–75, 1988. [Online]. Available : http://dx.doi.org/10.1007/BF00206326

[4] H. Corrigan-Gibbs and B. Ford, “Dissent : Accountable anonymous group messaging,” in *Proceedings of the 17th ACM Conference on Computer and Communications Security*, ser. CCS ’10. New York, NY, USA : ACM, 2010, pp. 340–350. [Online]. Available : http://doi.acm.org/10.1145/1866307.1866346

[5] D. I. Wolinsky, H. Corrigan-Gibbs, B. Ford, and A. Johnson, “Dissent in numbers : Making strong anonymity scale,” in *Presented as part of the 10th USENIX Symposium on Operating Systems Design and Implementation*. Berkeley, CA : USENIX, 2012, pp. 179–182. [Online]. Available : https://www.usenix.org/conference/osdi12/strong-scalable-anonymity-safetynet

[6] S. Goel, M. Robson, M. Polte, and E. Sirer, “Herbivore : A scalable and efficient protocol for anonymous communication. tech rep.” Cornell University, Tech. Rep., 2003.

[7] E. Adar and B. A. Huberman, “Free riding on gnutella,” *First Monday*, vol. 5, no. 10, 2000. [Online]. Available : http://firstmonday.org/ojs/index.php/fm/article/view/792/701&lt%3B/Hu96

[8] A. S. Aiyer, L. Alvisi, A. Clement, M. Dahlin, J.-P. Martin, and C. Porth, “Bar fault tolerance for cooperative services,” in *Proceedings of the Twentieth ACM Symposium on Operating Systems Principles*, ser. SOSP ’05. New York, NY, USA : ACM, 2005, pp. 45–58. [Online]. Available : http://doi.acm.org/10.1145/1095810.1095816

[9] H. C. Li, A. Clement, E. L. Wong, J. Napper, I. Roy, L. Alvisi, and M. Dahlin, “Bar gossip,” in *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, ser. OSDI ’06. Berkeley, CA,

USA : USENIX Association, 2006, pp. 191–204. [Online]. Available : http://dl.acm.org/citation.cfm?id=1298455.1298474

[10] S. Mokhtar, A. Pace, and V. Quema, "Firespam : Spam resilient gossiping in the bar model," in *Reliable Distributed Systems, 2010 29th IEEE Symposium on*, 2010, pp. 225–234.

[11] D. Hughes, G. Coulson, and J. Walkerdine, "Free riding on gnutella revisited : the bell tolls ?" *Distributed Systems Online, IEEE*, vol. 6, no. 6, pp. –, 2005.

[12] G. Hardin, "The tragedy of the commons. The population problem has no technical solution ; it requires a fundamental extension in morality." *Science*, vol. 162, no. 859, 1968.

[13] "The omnet++ simulation environment," 2013, http://www.omnetpp.org/.

[14] F. B. Schneider, "Implementing fault-tolerant services using the state machine approach : a tutorial," *ACM Comput. Surv.*, vol. 22, no. 4, pp. 299–319, 1990.

[15] X. Défago, A. Schiper, and P. Urbán, "Total order broadcast and multicast algorithms : Taxonomy and survey," *ACM Comput. Surv.*, vol. 36, no. 4, pp. 372–421, 2004.

[16] R. Guerraoui, R. R. Levy, B. Pochon, and V. Quéma, "Throughput optimal total order broadcast for cluster environments," *ACM Trans. Comput. Syst.*, vol. 28, no. 2, pp. 5 :1–5 :32, Jul. 2010. [Online]. Available : http://doi.acm.org/10.1145/1813654.1813656

[17] P. Marandi, M. Primi, N. Schiper, and F. Pedone, "Ring paxos : A high-throughput atomic broadcast protocol," in *Dependable Systems and Networks (DSN), 2010 IEEE/IFIP International Conference on*, 2010, pp. 527–536.

[18] J. B.-d. Siobhan Gorman, "New details show broader nsa surveillance reach," août 2013. [Online]. Available : http://online.wsj.com/article/SB10001424127887324108204579022874091732470.html?mod=WSJEurope_hpp_LEFTTopStories

[19] A. Pfitzmann and M. Hansen, "Anonymity, unlinkability, undetectability, unobservability, pseudonymity, and identity management – a consolidated proposal for terminology," 2008.

[20] J.-F. Raymond, "Traffic analysis : Protocols, attacks, design issues, and open problems," in *Designing Privacy Enhancing Technologies*, ser. Lecture Notes in Computer Science, H. Federrath, Ed. Springer Berlin Heidelberg, 2001, vol. 2009, pp. 10–29. [Online]. Available : http://dx.doi.org/10.1007/3-540-44702-4_2

[21] R. Newman, I. Moskowitz, P. Syverson, and A. Serjantov, "Metrics for traffic analysis prevention," in *Privacy Enhancing Technologies*, ser. Lecture Notes in Computer Science, R. Dingledine, Ed. Springer Berlin Heidelberg, 2003, vol. 2760, pp. 48–65. [Online]. Available : http://dx.doi.org/10.1007/978-3-540-40956-4_4

[22] D. L. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," *Commun. ACM*, vol. 24, no. 2, pp. 84–90, Feb. 1981. [Online]. Available : http://doi.acm.org/10.1145/358549.358563

[23] M. Waidner and B. Pfitzmann, "The dining cryptographers in the disco - underconditional sender and recipient untraceability with computationally

secure serviceability (abstract)," in *EUROCRYPT 1989*, 1989, p. 690. [Online]. Available : http://dx.doi.org/10.1007/3-540-46885-4_69

[24] P. Golle and A. Juels, "Dining cryptographers revisited," in *Advances in Cryptology - EUROCRYPT 2004*, ser. Lecture Notes in Computer Science, C. Cachin and J. Camenisch, Eds. Springer Berlin Heidelberg, 2004, vol. 3027, pp. 456–473. [Online]. Available : http://dx.doi.org/10.1007/978-3-540-24676-3_27

[25] S. Helmers, "A brief history of anon.penet.fi - the legendary anonymous remailer," September 1997. [Online]. Available : http://www.december.com/cmc/mag/1997/sep/helmers.html

[26] S. PAREKH, "Prospect for remailers," august 1996. [Online]. Available : http://firstmonday.org/ojs/index.php/fm/article/view/476/397

[27] G. Danezis, R. Dingledine, and N. Mathewson, "Mixminion : design of a type iii anonymous remailer protocol," in *Security and Privacy, 2003. Proceedings. 2003 Symposium on*, 2003, pp. 2–15.

[28] C. Gulcu and G. Tsudik, "Mixing e-mail with babel," in *Network and Distributed System Security, 1996., Proceedings of the Symposium on*, 1996, pp. 2–16.

[29] G. Danezis, "Mix-networks with restricted routes," in *Privacy Enhancing Technologies*, ser. Lecture Notes in Computer Science, R. Dingledine, Ed. Springer Berlin Heidelberg, 2003, vol. 2760, pp. 1–17. [Online]. Available : http://dx.doi.org/10.1007/978-3-540-40956-4_1

[30] O. Berthold, A. Pfitzmann, and R. Standtke, "The disadvantages of free mix routes and how to overcome them," in *Designing Privacy Enhancing Technologies*, ser. Lecture Notes in Computer Science, H. Federrath, Ed. Springer Berlin Heidelberg, 2001, vol. 2009, pp. 30–45. [Online]. Available : http://dx.doi.org/10.1007/3-540-44702-4_3

[31] R. Dingledine, V. Shmatikov, and P. Syverson, "Synchronous batching : From cascades to free routes," in *Privacy Enhancing Technologies*, ser. Lecture Notes in Computer Science, D. Martin and A. Serjantov, Eds. Springer Berlin Heidelberg, 2005, vol. 3424, pp. 186–206. [Online]. Available : http://dx.doi.org/10.1007/11423409_12

[32] A. Serjantov, R. Dingledine, and P. Syverson, "From a trickle to a flood : Active attacks on several mix types," in *Information Hiding*, ser. Lecture Notes in Computer Science, F. Petitcolas, Ed. Springer Berlin Heidelberg, 2003, vol. 2578, pp. 36–52. [Online]. Available : http://dx.doi.org/10.1007/3-540-36415-3_3

[33] C. Díaz and A. Serjantov, "Generalising mixes," in *Privacy Enhancing Technologies*, ser. Lecture Notes in Computer Science, R. Dingledine, Ed. Springer Berlin Heidelberg, 2003, vol. 2760, pp. 18–31. [Online]. Available : http://dx.doi.org/10.1007/978-3-540-40956-4_2

[34] R. N. A. Serjantov, "On the anonymity of timed pool mixes," in *Workshop on Privacy and Anonymity in Network and Distributed Systems*, 2003.

[35] D. Kedogan, D. Agrawal, and S. Penz, "Limits of anonymity in open environments," in *Information Hiding*, ser. Lecture Notes in Computer Science, F. Petitcolas, Ed. Springer Berlin Heidelberg, 2003, vol. 2578, pp. 53–69. [Online]. Available : http://dx.doi.org/10.1007/3-540-36415-3_4

[36] G. Danezis and L. Sassaman, "Heartbeat traffic to counter (n-1) attacks : red-green-black mixes," in *Proceedings of the 2003 ACM workshop on Privacy in the electronic society*, ser. WPES '03. New York, NY, USA : ACM, 2003, pp. 89–93. [Online]. Available : http://doi.acm.org/10.1145/1005140.1005154

[37] L. O'Connor, "On blending attacks for mixes with memory," in *Information Hiding*, ser. Lecture Notes in Computer Science, M. Barni, J. Herrera-Joancomartí, S. Katzenbeisser, and F. Pérez-González, Eds. Springer Berlin Heidelberg, 2005, vol. 3727, pp. 39–52. [Online]. Available : http://dx.doi.org/10.1007/11558859_4

[38] G. Danezis, "The traffic analysis of continuous-time mixes," in *Privacy Enhancing Technologies*, ser. Lecture Notes in Computer Science, D. Martin and A. Serjantov, Eds. Springer Berlin Heidelberg, 2005, vol. 3424, pp. 35–50. [Online]. Available : http://dx.doi.org/10.1007/11423409_3

[39] B. Levine, M. Reiter, C. Wang, and M. Wright, "Timing attacks in low-latency mix systems," in *Financial Cryptography*, ser. Lecture Notes in Computer Science, A. Juels, Ed. Springer Berlin Heidelberg, 2004, vol. 3110, pp. 251–265. [Online]. Available : http://dx.doi.org/10.1007/978-3-540-27809-2_25

[40] L. Overlier and P. Syverson, "Locating hidden servers," in *Security and Privacy, 2006 IEEE Symposium on*, 2006, pp. 15 pp.–114.

[41] A. Serjantov and P. Sewell, "Passive attack analysis for connection-based anonymity systems," in *Computer Security – ESORICS 2003*, ser. Lecture Notes in Computer Science, E. Snekkenes and D. Gollmann, Eds. Springer Berlin Heidelberg, 2003, vol. 2808, pp. 116–131. [Online]. Available : http://dx.doi.org/10.1007/978-3-540-39650-5_7

[42] X. Wang, S. Chen, and S. Jajodia, "Tracking anonymous peer-to-peer voip calls on the internet," in *Proceedings of the 12th ACM conference on Computer and communications security*, ser. CCS '05. New York, NY, USA : ACM, 2005, pp. 81–91. [Online]. Available : http://doi.acm.org/10.1145/1102120.1102133

[43] Y. Zhu and R. Bettati, "Un-mixing mix traffic," in *In Proceedings of Privacy Enhancing Technologies workshop (PET 2005)*, 2005.

[44] Y. Zhu, X. Fu, B. Graham, R. Bettati, and W. Zhao, "On flow correlation attacks and countermeasures in mix networks," in *Privacy Enhancing Technologies*, ser. Lecture Notes in Computer Science, D. Martin and A. Serjantov, Eds. Springer Berlin Heidelberg, 2005, vol. 3424, pp. 207–225. [Online]. Available : http://dx.doi.org/10.1007/11423409_13

[45] M. Reed, P. Syverson, and D. Goldschlag, "Anonymous connections and onion routing," *Selected Areas in Communications, IEEE Journal on*, vol. 16, no. 4, pp. 482–494, 1998.

[46] P. Syverson, G. Tsudik, M. Reed, and C. Landwehr, "Towards an analysis of onion routing security," in *Designing Privacy Enhancing Technologies*, ser. Lecture Notes in Computer Science, H. Federrath, Ed. Springer Berlin Heidelberg, 2001, vol. 2009, pp. 96–114. [Online]. Available : http://dx.doi.org/10.1007/3-540-44702-4_6

[47] A. Back, U. Möller, and A. Stiglic, "Traffic analysis attacks and trade-offs in anonymity providing systems," in *Information Hiding*, ser.

Lecture Notes in Computer Science, I. Moskowitz, Ed. Springer Berlin Heidelberg, 2001, vol. 2137, pp. 245–257. [Online]. Available : http://dx.doi.org/10.1007/3-540-45496-9_18

[48] R. Dingledine and N. Mathewson, "Anonymity loves company : Usability and the network effect," in *Proceedings of the Fifth Workshop on the Economics of Information Security (WEIS 2006)*, 2006. [Online]. Available : http://weis2006.econinfosec.org/docs/41.pdf

[49] R. Dingledine, N. Mathewson, and P. Syverson, "Deploying low-latency anonymity : Design challenges and social factors," *Security Privacy, IEEE*, vol. 5, no. 5, pp. 83–87, 2007.

[50] A. Blum, D. Song, and S. Venkataraman, "Detection of interactive stepping stones : Algorithms and confidence bounds," in *Recent Advances in Intrusion Detection*, ser. Lecture Notes in Computer Science, E. Jonsson, A. Valdes, and M. Almgren, Eds. Springer Berlin Heidelberg, 2004, vol. 3224, pp. 258–277. [Online]. Available : http://dx.doi.org/10.1007/978-3-540-30143-1_14

[51] X. Wang and D. S. Reeves, "Robust correlation of encrypted attack traffic through stepping stones by manipulation of interpacket delays," in *Proceedings of the 10th ACM conference on Computer and communications security*, ser. CCS '03. New York, NY, USA : ACM, 2003, pp. 20–29. [Online]. Available : http://doi.acm.org/10.1145/948109.948115

[52] M. J. Freedman and R. Morris, "Tarzan : A peer-to-peer anonymizing network layer," in *Proceedings of the 9th ACM Conference on Computer and Communications Security*, ser. CCS '02. New York, NY, USA : ACM, 2002, pp. 193–206. [Online]. Available : http://doi.acm.org/10.1145/586110.586137

[53] M. Rennhard and B. Plattner, "Introducing morphmix : peer-to-peer based anonymous internet usage with collusion detection," in *Proceedings of the 2002 ACM workshop on Privacy in the Electronic Society*, ser. WPES '02. New York, NY, USA : ACM, 2002, pp. 91–102. [Online]. Available : http://doi.acm.org/10.1145/644527.644537

[54] G. Ciaccio, "Improving sender anonymity in a structured overlay with imprecise routing," in *Privacy Enhancing Technologies*, ser. Lecture Notes in Computer Science, G. Danezis and P. Golle, Eds. Springer Berlin Heidelberg, 2006, vol. 4258, pp. 190–207. [Online]. Available : http://dx.doi.org/10.1007/11957454_11

[55] A. Nambiar and M. Wright, "Salsa : a structured approach to large-scale anonymity," in *Proceedings of the 13th ACM conference on Computer and communications security*, ser. CCS '06. New York, NY, USA : ACM, 2006, pp. 17–26. [Online]. Available : http://doi.acm.org/10.1145/1180405.1180409

[56] A. Mislove, G. Oberoi, A. Post, C. Reis, P. Druschel, and D. S. Wallach, "Ap3 : cooperative, decentralized anonymous communication," in *Proceedings of the 11th workshop on ACM SIGOPS European workshop*, ser. EW 11. New York, NY, USA : ACM, 2004. [Online]. Available : http://doi.acm.org/10.1145/1133572.1133578

[57] L. Zhuang, F. Zhou, B. Y. Zhao, and A. Rowstron, "Cashmere : Resilient anonymous routing," in *Proceedings of the 2Nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2*, ser. NSDI'05. Berkeley, CA, USA : USENIX Association, 2005, pp. 301–314. [Online]. Available : http://dl.acm.org/citation.cfm?id=1251203.1251225

[58] F. Li, B. Luo, P. Liu, and C.-H. Chu, "A node-failure-resilient anonymous communication protocol through commutative path hopping," in *INFOCOM, 2010 Proceedings IEEE*, 2010, pp. 1–9.

[59] Y. Zhu and Y. Hu, "Making peer-to-peer anonymous routing resilient to failures," in *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, 2007, pp. 1–10.

[60] K. Puttaswamy, A. Sala, C. Wilson, and B. Zhao, "Protecting anonymity in dynamic peer-to-peer networks," in *Network Protocols, 2008. ICNP 2008. IEEE International Conference on*, 2008, pp. 104–113.

[61] A. J. Jansen, R. and P. Syverson, "Lira : Lightweight incentivized routing for anonymity," in *20th Annual Network Distributed System Security Symposium (NDSS '13)*, 2013.

[62] R. Jansen, P. Syverson, and N. Hopper, "Throttling tor bandwidth parasites," in *Proceedings of the 21st USENIX conference on Security symposium*, ser. Security'12. Berkeley, CA, USA : USENIX Association, 2012, pp. 18–18. [Online]. Available : http://dl.acm.org/citation.cfm?id=2362793.2362811

[63] R. Sherwood, B. Bhattacharjee, and A. Srinivasan, "P5 : A protocol for scalable anonymous communication," in *IEEE Symposium on Security and Privacy*, 2002, pp. 58–70. [Online]. Available : http://doi.ieeecomputersociety.org/10.1109/SECPRI.2002.1004362

[64] H. Johansen, A. Allavena, and R. van Renesse, "Fireflies : Scalable support for intrusion-tolerant network overlays," in *Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems 2006*, ser. EuroSys '06. New York, NY, USA : ACM, 2006, pp. 3–13. [Online]. Available : http://doi.acm.org/10.1145/1217935.1217937

[65] A.-M. Kermarrec, L. Massoulie, and A. Ganesh, "Probabilistic reliable dissemination in large-scale systems," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 14, no. 3, pp. 248–258, 2003.

[66] R. Pries, W. Yu, X. Fu, and W. Zhao, "A new replay attack against anonymous communication networks," in *Communications, 2008. ICC '08. IEEE International Conference on*, 2008, pp. 1578–1582.

[67] J. Nash, "Non-Cooperative Games," *Annals of Mathematics*, vol. 54, no. 2, 1951.

[68] A. Varga and R. Hornig, "An overview of the omnet++ simulation environment," in *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*, ser. Simutools '08. ICST, Brussels, Belgium, Belgium : ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008, pp. 60 :1–60 :10. [Online]. Available : http://dl.acm.org/citation.cfm?id=1416222.1416290

[69] E. Weingartner, H. vom Lehn, and K. Wehrle, "A performance comparison of recent network simulators," in *Communications, 2009. ICC '09. IEEE International Conference on*, 2009, pp. 1–5.

[70] I. Rüngeler, M. Tüxen, and E. P. Rathgeb, "Congestion and flow control in the context of the message-oriented protocol sctp," in *Proceedings of the 8th International IFIP-TC 6 Networking Conference*, ser. NETWORKING '09. Berlin, Heidelberg : Springer-Verlag, 2009, pp. 468–481. [Online]. Available : http://dx.doi.org/10.1007/978-3-642-01399-7_37

[71] M. Kozlovszky, A. Balasko, and A. Varga, "Enabling omnet++-based simulations on grid systems," in *Proceedings of the 2Nd International Conference on Simulation Tools and Techniques*, ser. Simutools '09. ICST, Brussels, Belgium, Belgium : ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2009, pp. 67 :1–67 :7. [Online]. Available : http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5569

[72] V. Hadzilacos and S. Toueg, "Distributed systems (2nd ed.)," S. Mullender, Ed. New York, NY, USA : ACM Press/Addison-Wesley Publishing Co., 1993, ch. Fault-tolerant Broadcasts and Related Problems, pp. 97–145. [Online]. Available : http://dl.acm.org/citation.cfm?id=302430.302435

[73] F. Kaashoek and A. Tanenbaum, "An evaluation of the amoeba group communication system," in *Proceedings of the 16th International Conference on Distributed Computing Systems (ICDCS '96)*. Washington, DC, USA : IEEE Computer Society, 1996.

[74] S. Armstrong, A. Freier, and K. Marzullo, "Multicast transport protocol," RFC 1301, IETF, 1992.

[75] R. Carr, "The tandem global update protocol," *Tandem Syst. Rev. 1*, pp. 74–85, jun 1985.

[76] H. Garcia-Molina and A. Spauster, "Ordered and reliable multicast communication," *ACM Trans. Comput. Syst.*, vol. 9, no. 3, pp. 242–271, 1991.

[77] K. Birman and R. van Renesse, *Reliable Distributed Computing with the Isis Toolkit*. IEEE Computer Society Press, 1993.

[78] U. Wilhelm and A. Schiper, "A hierarchy of totally ordered multicasts," in *Proceedings of the 14TH Symposium on Reliable Distributed Systems*. Washington, DC, USA : IEEE Computer Society, 1995.

[79] P. Marandi, M. Primi, and F. Pedone, "Multi-ring paxos," in *Dependable Systems and Networks (DSN), 2012 42nd Annual IEEE/IFIP International Conference on*, 2012, pp. 1–12.

[80] N. A. Lynch, *Distributed Algorithms*. Morgan-Kaufmann, 1996.

[81] T. Chandra and S. Toueg, "Unreliable failure detectors for reliable distributed systems," *J. ACM*, vol. 43, no. 2, pp. 225–267, 1996.

[82] J. Dunagan, N. J. A. Harvey, M. B. Jones, D. Kostic, M. Theimer, and A. Wolman, "Fuse : Lightweight guaranteed distributed failure notification," in *Proceedings of 6th Symposium on Operating Systems Design and Implementation (OSDI '04)*, 2004.

[83] K. Birman and T. Joseph, "Exploiting virtual synchrony in distributed systems," in *Proceedings of the eleventh ACM Symposium on Operating systems principles (SOSP'87)*. New York, NY, USA : ACM Press, 1987, pp. 123–138.

[84] R. Baldoni, S. Cimmino, and C. Marchetti, "A Classification of Total Order Specifications and its Application to Fixed Sequencer-based Implementations," *to appear in Journal of Parallel and Distributed Computing*, 6 2006.

[85] X. Défago, A. Schiper, and P. Urbán, "Comparative performance analysis of ordering strategies in atomic broadcast algorithms," *IEICE Trans. on Information and Systems*, vol. E86-D, no. 12, pp. 2698–2709, 2003.

[86] J.-M. Chang and N. Maxemchuk, "Reliable broadcast protocols," *ACM Trans. Comput. Syst.*, vol. 2, no. 3, pp. 251–273, 1984.

[87] B. Whetten, T. Montgomery, and S. Kaplan, "A high performance totally ordered multicast protocol," in *Selected Papers from the International Workshop on Theory and Practice in Distributed Systems*. London, UK : Springer-Verlag, 1994, pp. 33–57.

[88] J. Kim and C. Kim, "A total ordering protocol using a dynamic token-passing scheme," *Distrib. Syst. Eng. J.*, vol. 4, no. 2, pp. 87–95, jun 1997.

[89] F. Cristian, S. Mishra, and G. Alvarez, "High-performance asynchronous atomic broadcast," *Distrib. Syst. Eng. J.*, vol. 4, no. 2, pp. 109–128, jun 1997.

[90] T. Friedman and R. V. Renesse, "Packing messages as a tool for boosting the performance of total ordering protocls," in *Proceedings of the 6th International Symposium on High Performance Distributed Computing (HPDC '97)*. Washington, DC, USA : IEEE Computer Society, 1997.

[91] F. Cristian, "Asynchronous atomic broadcast," *IBM Technical Disclosure Bulletin*, vol. 33, no. 9, pp. 115–116, 1991.

[92] R. Ekwall, A. Schiper, and P. Urban, "Token-based atomic broadcast using unreliable failure detectors," in *Proceedings of the 23rd IEEE International Symposium on Reliable Distributed Systems (SRDS'04)*. Washington, DC, USA : IEEE Computer Society, 2004, pp. 52–65.

[93] Y. Amir, L. E. Moser, P. M. Melliar-Smith, D. A. Agarwal, and P. Ciarfella, "The Totem single-ring ordering and membership protocol," *ACM Transactions on Computer Systems*, vol. 13, no. 4, pp. 311–342, 1995.

[94] A. Gopal and S. Toueg, "Reliable broadcast in synchronous and asynchronous environments (preliminary version)," in *Proceedings of the 3rd International Workshop on Distributed Algorithms*. London, UK : Springer-Verlag, 1989, pp. 110–123.

[95] L. Peterson, N. Buchholz, and R. Schlichting, "Preserving and using context information in interprocess communication," *ACM Trans. Comput. Syst.*, vol. 7, no. 3, pp. 217–246, 1989.

[96] L. Malhis, W. Sanders, and R. Schlichting, "Numerical performability evaluation of a group multicast protocol," *Distrib. Syst. Enj. J.*, vol. 3, no. 1, pp. 39–52, march 1996.

[97] P. Ezhilchelvan, R. Macedo, and S. Shrivastava, "Newtop : a fault-tolerant group communication protocol," in *Proceedings of the 15th International*

Conference on Distributed Computing Systems (ICDCS'95). Washington, DC, USA : IEEE Computer Society, 1995.

[98] T. Ng, "Ordered broadcasts for large applications," in *Proceedings of the 10th IEEE International Symposium on Reliable Distributed Systems (SRDS'91)*. Pisa, Italy : IEEE Computer Society, 1991, pp. 188–197.

[99] L. Moser, P. Melliar-Smith, and V. Agrawala, "Asynchronous fault-tolerant total ordering algorithms," *SIAM J. Comput.*, vol. 22, no. 4, pp. 727–750, 1993.

[100] K. Birman and T. Joseph, "Reliable communication in the presence of failures," *ACM Trans. Comput. Syst.*, vol. 5, no. 1, pp. 47–76, 1987.

[101] S. Luan and V. Gligor, "A fault-tolerant protocol for atomic broadcast," *IEEE Trans. Parallel Distrib. Syst.*, vol. 1, no. 3, pp. 271–285, 1990.

[102] U. Fritzke, P. Ingels, A. Mostefaoui, and M. Raynal, "Consensus-based fault-tolerant total order multicast," *IEEE Trans. Parallel Distrib. Syst.*, vol. 12, no. 2, pp. 147–156, 2001.

[103] E. Anceaume, "A lightweight solution to uniform atomic broadcast for asynchronous systems," in *Proceedings of the 27th International Symposium on Fault-Tolerant Computing (FTCS '97)*. Washington, DC, USA : IEEE Computer Society, 1997.

[104] L. Lamport, "Paxos made simple," *ACM Sigact News*, vol. 32, no. 4, pp. 18–25, 2001.

[105] J.-Y. Le Boudec, "Rate adaptation, congestion control and fairness : A tutorial," Ecole Polytechnique Fédérale de Lausanne, 2012.

[106] R. Jones, *Netperf*, http ://www.netperf.org/, 2007.

Table des figures

Liste des tableaux

Résumé

Cette thèse porte sur la transmission d'informations dans les réseaux d'ordinateurs. Nous nous sommes plus particulièrement penchés sur deux aspects de ce problème : les communications anonymes sur Internet en présence de nœuds rationnels (aussi appelés "égoïstes") et la diffusion à ordre uniformément total dans le cadre d'une grappe de machines. Concernant le premier aspect, nous avons constaté qu'il n'existait pas de protocole de communications anonymes fonctionnant en présence de nœuds rationnels et capable de monter en charge (c'est-à-dire de fonctionner efficacement en présence d'un grand nombre de nœuds). Nous avons donc proposé *RAC*, le premier protocole de communications anonymes capable de monter en charge et fonctionnant en présence de nœuds rationnels. Concernant le deuxième aspect, nous avons constaté qu'il n'existait pas de protocole de diffusion à ordre uniformément total assurant à la fois un débit optimal et une latence faible. Nous avons donc proposé *FastCast*, le premier protocole de diffusion à ordre uniformément total garantissant un débit optimal tout en assurant une latence faible.

Mots-clés. Communications anonymes, systèmes pair-à-pair, montée en charge, diffusion à ordre uniformément total, multi-diffusion IP, efficacité en débit, efficacité en latence.

Abstract

This thesis focuses on information dissemination in computer networks. We study two aspects of this topic : anonymous communication on Internet with rational nodes and uniform total order broadcast in a computer cluster. Concerning the first aspect, we observed that no anonymous communication protocol is capable of working with rational nodes while scaling existed. Therefore, we proposed *RAC*, the first anonymous communication protocol functioning with rational nodes and able of scaling. Concerning the second aspect, we observed that no existing uniform total order broadcast protocol is capable of ensuring both a good latency and an optimal throughput. In order to fill this lack we proposed *FastCast*, the first uniform total order broadcast ensuring both an optimal throughput and a low latency.

Keywords. Anonymous communications, peer-to-peer systems, scaling, uniform total order broadcast, IP multicast, throughput-efficiency, latency-efficiency.