Diffusion en pair à pair – Sujet Corrigé

1 Diffusion d'un fichier dans un environnement pair-à-pair

On considère un réseau pair-à-pair à n pairs ; un fichier est possédé par une source, et on cherche à le diffuser à tous les autres nœuds. Le temps de diffusion du fichier entier est 1. Un nœud ne peut envoyer qu'un fichier (ou bout de fichier) à la fois.

Question 1.1. *On considère l'algorithme simple suivant : dès qu'un nœud reçoit le fichier, et tant que tout le monde n'a pas reçu sa copie, le nœud envoie le fichier à un autre nœud qui ne le possède pas encore. Quel est le temps requis par cet algorithme pour diffuser le fichier ?*

Correction Solution : $\lceil \log_2 n \rceil$ (à chaque étape, le nombre de nœuds possédant le fichier a doublé). □

Question 1.2. *On découpe maintenant le fichier en b blocs. Le temps de diffusion d'un bloc est donc $1/b$. Donner une borne inférieure sur le temps de diffusion du fichier complet.*

Correction Solution : La source aura donné une copie de chaque bloc au bout d'un temps 1. Le dernier bloc sera donc possédé par seulement 2 nœuds au temps 1.
Le transfert d'une copie de ce bloc prend un temps $1/b$, donc le fichier sera possédé par toutes les pairs au mieux au bout d'un temps $(\lceil \log_2 n \rceil - 1)/b$.
D'où la borne inf :

$$1 + \frac{\lceil \log_2 n \rceil - 1}{b}$$

□

On suppose maintenant que chaque nœud peut recevoir et envoyer un bloc tous les $1/b$, et que $n = 2^l$. Les nœuds sont numérotés de 0 à $n-1$ (la source a le numéro $s = 0$). On désigne par $a \lll k$ l'opération de décalage circulaire des bits du numéro a de k positions vers la gauche (modulo l), et par $\oplus$ le ou exclusif bit à bit (XOR). On propose alors l'algorithme de diffusion suivant :

1. La source donne le bloc i au temps i/b au nœud $1 \lll i$
2. Au temps t/b, chaque nœud u redonne le dernier bloc reçu au nœud $u \oplus (1 \lll t)$.

Question 1.3.

(a) Donner l'ensemble des nœuds possédant le bloc i au temps $(i + k)/b$ pour $k = 0, 1, 2 \ldots$. En déduire le nombre d'étapes nécessaires à la diffusion du bloc i, et le temps d'exécution de l'algorithme. Est-ce optimal ? Pourquoi ?

(b) Montrer qu'il n'y a pas de conflit pour des communications simultanées. Quelle est le graphe de communication utilisé dans cet algorithme ?

(c) Proposer une variante lorsque n n'est pas une puissance de 2.

(d) Cet algorithme est-il utilisable en pratique ?

Correction Voir l'HDR de Laurent Viennot pp. 44-45.

(a) Le bloc i est possédé :
- par la source au temps i/b
- par le nœud $1 \lll i$ au temps $(i+1)/b$
- par les nœuds $1 \lll i$ et $11 \lll i$ au temps $(i+2)/b$
- ...
- par les nœuds $b_1 \dots b_k 1 \lll i$ au temps $(i+k+1)/b$
- par tous les nœuds au temps $(i+l)/b$ (lors de la dernière étape, tous les nœuds $b_1 \dots b_{l-1} 1 \lll i$ donne le bloc aux nœuds $b_1 \dots b_{l-1} 0 \lll i$).

Donc le temps d'exécution de l'algorithme est $1 + l/b$, ce qui est presque optimal. Pour atteindre l'optimal, il faut que la source participe à la diffusion du dernier bloc (sinon elle ne fait rien à partir de $t = b$).

(b) Les communications s'organisent par paire, il n'y a jamais 2 nœuds qui envoient un message au même nœud. Le graphe des communications est un hypercube On pourra faire un exemple avec $n = 8$, et tracer les communications impliquées par la diffusion des bloc 0, 1,2. . . ils doivent former des arbres arêtes disjoints enracinés en 1,2,4. . .

(c) On peut regrouper les nœud en groupe de puissance de 2, ou bien faire ça sur les $2^{\lfloor log_2 n \rfloor}$, et rajouter une dernière étape pour diffuser aux nœuds de plus grand numéros.

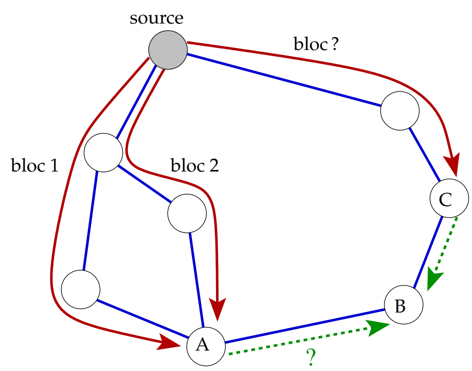
(d) Pas vraiment, il suppose une grande fiabilité des pairs et des capacités de communications homogènes.

□

2 Diffusion de blocs dans Avalanche : le Network Coding

Des protocoles pair-à-pair comme BitTorrent utilisent le découpage en blocs, en utilisant un contrôle distribué pour savoir quel bloc envoyer à quel pair. Cependant, ce n'est parfois pas suffisant pour éviter que certains blocs se raréfient à cause des déconnections. On illustre ici une technique pour éviter ce problème.

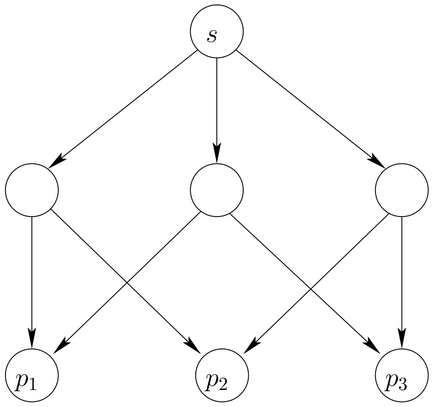
Question 2.1. *Sur le schéma ci-contre, le message à transmettre est constitué de deux blocs, B_1 et B_2. Le nœud A souhaite transmettre un bloc à B, mais il ignore quel bloc le nœud B va recevoir de C. Comment faire pour lui transmettre un seul bloc lui permettant de reconstituer l'intégralité du message ?*

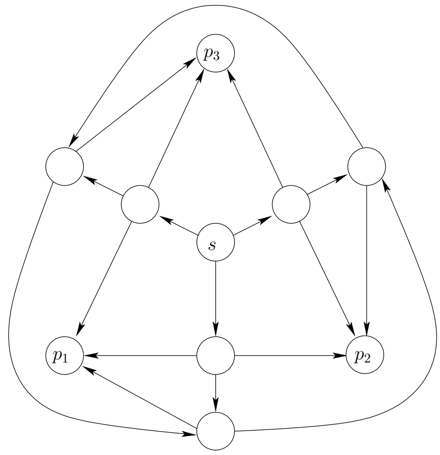


Correction Il suffit que le nœud A envoie le message $B_A = B_1 \oplus B_2$: quelque soit l'autre bloc B_B reçu du nœud B, on pourra reconstituer le bloc manquant en calculant $B_A \oplus B_B$ ($\oplus$ = XOR). □

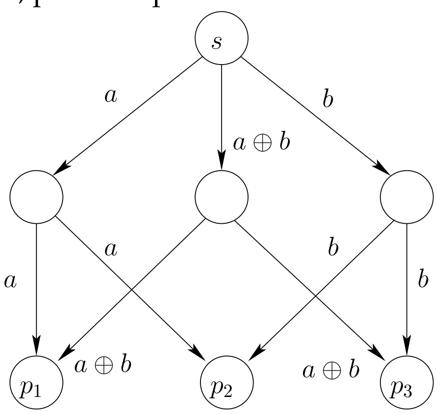
À l'origine, cette technique (appelée Network Coding) permet de multiplexer des flux pour augmenter le débit dans un réseau.

Question 2.2. *Dans les réseaux suivants, où toutes les capacités des liens valent 1, calculer le flot maximum qu'on peut obtenir depuis la source vers chacun des puits p_1, p_2 et p_3 de façon indépendante. Quel débit peut-on obtenir dans les réseaux suivants avec et sans Network Coding ?*
(toutes les capacité des arêtes sont 1)

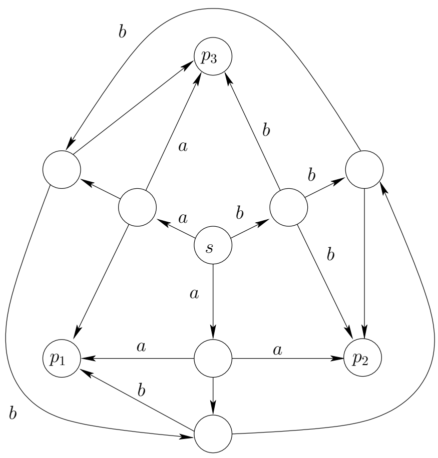




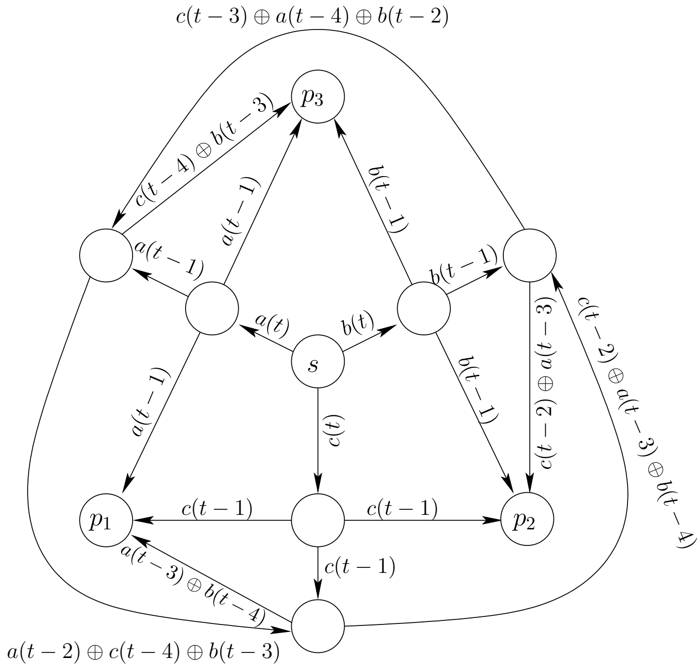
Correction Dans le réseau de gauche, sans network coding, on peut atteindre un débit de 3/2 au maximum (3 messages diffusés toutes les 2 unités de temps). Avec network coding, on peut atteindre un débit de 2, par exemple avec l'allocation suivante ($\oplus = XOR$) :



Dans le réseau de droite, sans network coding on peut atteindre un débit de 2, et avec un débit de 3 (voir ci-dessous).



sans network coding



avec network coding

La différence avec l'exemple précédent est qu'ici on a besoin d'entrelacer des versions différentes de chaque message : les messages du flux a sont donc étiquetés par l'instant t auquel ils ont été émis par la source. □

Ahlswede [1], puis Koetter [4], ont montré que pour tout graphe et tout couple source–destination, il existe un codage permettant d'atteindre le flot maximal. Ils proposent des algorithmes polynomiaux pour trouver des combinaisons permettant d'atteindre le flot maximal f pour r destinations, en travaillant dans $\mathbb{Z}/q\mathbb{Z}$, avec $q = f \times r$.

Cependant, on cherche plutôt à utiliser cette technique dans un contexte décentralisé. Des travaux [5, 3] montrent que si on choisit des combinaisons linéaires aléatoires (dans $\mathbb{Z}/q\mathbb{Z}$) dans un graphe à m arêtes, on peut reconstruire tous les blocs avec une probabilité supérieure à $1 - \dfrac{m \times r}{q}$.

C'est cette technique, utilisée dans le protocole pair-à-pair Avalanche [2] développé par Microsoft, que nous allons étudier ici.

Un fichier F est découpé en n blocs $B_1, \ldots B_n$. Chaque bloc B_i est découpé en k entiers positifs : $B_i = (x_{i,1}, \ldots, x_{i,k})$. Pour effectuer les opérations de combinaison, on se donne un grand nombre premier q de plusieurs centaines de bits (chaque $x_{i,j}$ est supposé strictement inférieur à q). La source fournit des combinaisons aléatoires

$$a_1B_1+\cdots+a_nB_n = \big((a_1x_{1,1}+\cdots+a_nx_{n,1})\,,\ (a_1x_{1,2}+\cdots+a_nx_{n,2})\,,\ \ldots\ ,\ (a_1x_{1,k}+\cdots+a_nx_{n,k})\big)$$

Les a_j sont tirés aléatoirement et les opérations sont effectuées modulo q (on travaille dans un corps puisque q est premier). Les coefficients a_j sont transmis en même temps que la combinaison, ce qui induit un certain surcoût de communication.

Vérification des combinaisons Une des difficulté du protocole réside dans la vérification de l'intégrité d'une combinaison : connaissant $a_1, \dots a_n$, comment vérifier que la combinaison linéaire $C = (y_1, \dots, y_k)$ qu'on a reçu est bien égale à $a_1 B_1 + \cdots + a_n B_n$? (Il faut pouvoir se protéger d'un client malicieux ou bogué qui introduit de mauvaises combinaisons). Pour cela, le protocole propose d'utiliser une fonction de hachage h homomorphe, c'est-à-dire telle que $h(a_1 B_1 + \cdots + a_n B_n) = h(B_1)^{a_1} \times \cdots \times h(B_n)^{a_n}$. Ainsi la connaissance de $h(B_1), \cdots, h(B_n)$ (transmis par la source) suffit pour vérifier l'intégrité de toute combinaison.

On utilise la fonction de hachage obtenue en trouvant deux nombre premiers p et q tels que q divise $p-1$ ($p-1 = d \times q$), en se donnant k nombres aléatoires $g_1, \dots, g_k$ et en posant :

$$h(y_1, \dots, y_k) = g_1^{dy_1} \times \cdots \times g_k^{dy_k}$$

On prend pour les g_i des nombres aléatoires de même longueur que p.

Question 2.3. *Montrer que h est un homomorphisme.*

NB. On pourra utiliser le petit théorème de Fermat : Pour tous entiers $g \neq 0$ et p premiers,

$$g^{p-1} = 1 \bmod p.$$

Correction Avec

$$y_j = a_1 x_{1,j} + \cdots + a_n x_{n,j} \bmod q = a_1 x_{1,j} + \cdots + a_n x_{n,j} + m_j q$$

pour un certain m_j. On a

$$\begin{aligned} h(a_1 B_1 + \cdots + a_n B_n) &= h(y_1, \dots y_n) \\ &= g_1^{dy_1} \times \cdots \times g_n^{dy_n} \end{aligned}$$

Comme $p - 1 = d \times q$, on a aussi

$$dy_j = d(a_1 x_{1,j} + \cdots + a_n x_{n,j}) + m_j dq = d(a_1 x_{1,j} + \cdots + a_n x_{n,j}) + m_j (p-1)$$

Et comme $g_j^{m_j(p-1)} = 1 \bmod p$,

$$g_j^{dy_j} = g_j^{d(a_1 x_{1,j} + \cdots + a_n x_{n,j})} \bmod p$$

et

$$h(a_1 B_1 + \cdots + a_n B_n) = h(B_1)^{a_1} \times \cdots \times h(B_n)^{a_n} \bmod p$$

Pb : on n'a le résultat que modulo p, mais apparemment ça suffit.

NB La sécurité de la fonction de hachage repose sur la difficulté de calculer des logarithmes discrets. (?) □

Question 2.4. *Une fois obtenues n combinaisons $C_1, \dots C_n$, comment peut-on reconstruire les blocs initiaux ? Quelle hypothèse doit-on faire ?*

Correction On note $C_i = a_{i,1}B_1 + \ldots + a_{i,n}B_n$. Alors, en considérant la matrice A des coefficients ainsi que les matrices B et C des blocs originaux ou reçus (chaque ligne représente un bloc) :

$$A = \begin{bmatrix} a_{1,1} & \ldots & a_{1,n} \\ \vdots & & \vdots \\ a_{i,1} & \ldots & a_{i,n} \\ \vdots & & \vdots \\ a_{n,1} & \ldots & a_{n,n} \end{bmatrix} \quad B = \begin{bmatrix} x_{1,1} & \ldots & x_{1,k} \\ \vdots & & \vdots \\ x_{i,1} & \ldots & x_{i,k} \\ \vdots & & \vdots \\ x_{n,1} & \ldots & x_{n,k} \end{bmatrix} \quad C = \begin{bmatrix} y_{1,1} & \ldots & y_{1,k} \\ \vdots & & \vdots \\ y_{i,1} & \ldots & y_{i,k} \\ \vdots & & \vdots \\ y_{n,1} & \ldots & y_{n,k} \end{bmatrix},$$

on a $C = A \times B$. On connaît A et C ; pour retrouver B, il faut calculer l'inverse de A, sous réserve que cette matrice soit inversible. □

Question 2.5. *Que faire si cette hypothèse n'est pas vérifiée ?*

Correction On demande un bloc de plus, jusqu'à ce que A soit de rang n. Néanmoins, une matrice à coefficients aléatoires est inversible avec forte probabilité, donc ce ne devrait pas se produire souvent. □

Question 2.6. *Calculer la complexité des opérations de décodage et de vérifications. Voyez vous un inconvénient à ce protocole ?*

Correction
- Pour le décodage : la complexité de l'inversion d'une matrice (par le pivot de Gauss) est $O(2/3n^3)$.
- Pour la vérification des coefficients, il faut calculer $h(C_i)$, puis vérifier que c'est égal à $h(B_1)^{a_1} \times \cdots \times h(B_n)^{a_n}$, d'où $(k+n)$ calcul de puissances (et $(k+n)$ multiplications), complexité : ?

Il ne faut pas oublier qu'on travaille avec des grands entiers (modulo q). □

Sources et références

[1] Rudolf Ahlswede, Ning Cai, Shuo-Yen Robert Li, and Raymond W. Yeung. Network information flow. *IEEE Transactions on Information Theory*, 46(4) :1204–1216, 2000.

[2] Christos Gkantsidis and Pablo Rodriguez Rodriguez. Network coding for large scale content distribution. In *INFOCOM*, 2005.

[3] T. Ho, D. Karger, M. Medard, and R. Koetter. Network coding from a network flow perspective, 2003.

[4] Tracey Ho, Muriel Médard, and Ralf Koetter. An information theoretic view of network management. In *INFOCOM*, 2003.

[5] Sidharth Jaggi, Peter Sanders, Philip A. Chou, Michelle Effros, Sebastian Egner, Kamal Jain, and Ludo M. G. M. Tolhuizen. Polynomial time algorithms for multicast network code construction. *IEEE Transactions on Information Theory*, 51(6) :1973–1982, 2005.