Diffusion en pair à pair

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 ?*

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.*

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 $u \oplus (1 \lll t)$.

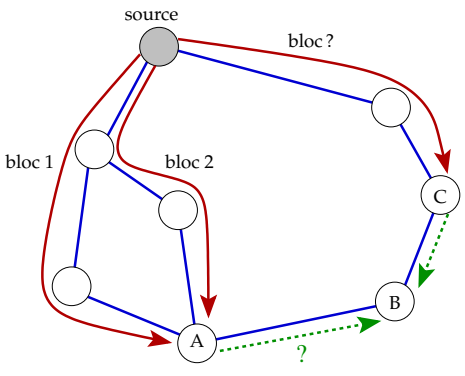
Question 1.3.

(a) Analyser cet algorithme, et montrer qu'il est optimal (au sens de la borne de la question 2).
(b) Proposer une variante lorsque n n'est pas une puissance de 2.
(c) Cet algorithme est-il utilisable en pratique ?

2 Diffusion de blocs dans Avalanche : le Network Coding

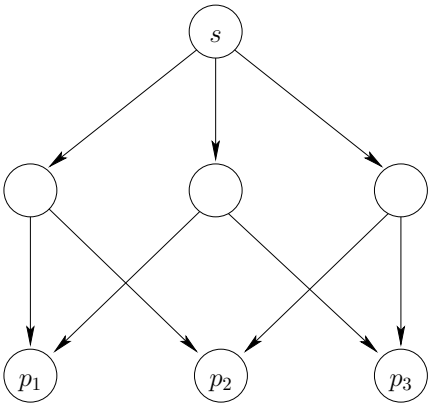
Des protocoles pair-à-pair comme BitTorrent utilisent le découpage en blocs, en utilisant un contrôle distribué pour savoir quel bloc envoyer à quel pair. Cependant, ce n'est parfois pas suffisant pour éviter que certains blocs se raréfient à cause des déconnections. On illustre ici une technique pour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

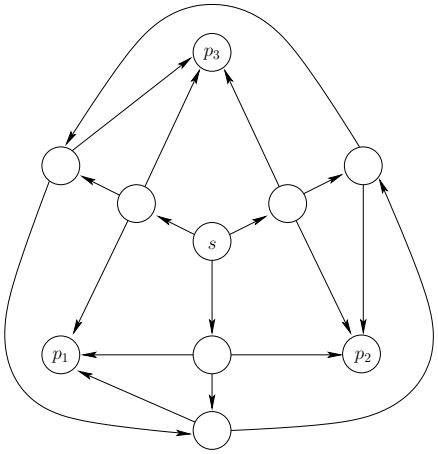
Question 2.1. *Sur le schéma ci-contre, le message à transmettre est constitué de deux blocs, B_1 et B_2. Le nœud A souhaite transmettre un bloc à B, mais il ignore quel bloc le nœud B va recevoir de C. Comment faire pour lui transmettre un seul bloc lui permettant de reconstituer l'intégralité du message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



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)





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 $1 - \frac{m \times r}{q}$.
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 $B_1, \dots B_n$. Chaque bloc B_i est découpé en k entiers positifs : $B_i = (x_{i,1}, \dots, x_{i,n})$. 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

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

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 $a_1B_1 + \dots + a_nB_n$? (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 $h(a_1B_1 + \cdots + a_nB_n) = h(B_1)^{a_1}h(B_n)^{a_n}$. Ainsi la connaissance de $h(B_1), \cdots, h(B_n)$ 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 $p - 1 = dq$ (q divise $p - 1$), en se donnant k nombres aléatoires $g_1, \ldots, g_k$ et en posant :

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

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

Question 2.3. *Montrer que h est un homomorphisme. On pourra utiliser le petit théorème de Fermat : Pour tous entiers $g \neq 0$ et p premiers, on a*

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

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

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

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

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.