# Recherche en temps réel de préfixes massifs hiérarchiques dans un réseau IP à l'aide de techniques de stream mining

Pascal Cheung-Mon-Chan*, Fabrice Clérot*

* France Télécom R&D
2, avenue Pierre Marzin BP 50702
22307 Lannion Cedex -France
{pascal.cheungmonchan, fabrice.clerot}@francetelecom.com

**Résumé.** Au cours de ces dernières années, de nombreuses techniques de stream mining ont été proposées afin d'analyser des flux de données en temps réel. Dans cet article, nous montrons comment nous avons utilisé des techniques de stream mining permettant la recherche d'objets massifs hiérarchiques (hierarchical heavy hitters) dans un flux de données pour identifier en temps réel dans un réseau IP les préfixes dont la contribution au trafic dépasse une certaine proportion de ce trafic pendant un intervalle de temps donné.

## 1 Introduction

Les progrès techniques récents ont eu pour conséquence l'augmentation du nombre de flux d'information et la croissance rapide de leurs débits. L'architecture traditionnelle de l'analyse de données — où les données, préalablement stockées, sont analysées puis rafraîchies — étant inadaptée au traitement de ces flux, une nouvelle famille de techniques, dites de stream mining, se propose d'inverser radicalement cette architecture et de mettre en oeuvre des systèmes reposant sur des capacités de stockage minimales qui sont mises à jour à la vitesse du flux. L'objectif de cet article est d'expliquer comment nous avons utilisé des techniques de stream mining afin d'identifier en temps réel, dans un réseau IP, les préfixes dont la contribution au trafic dépasse une certaine proportion de ce trafic pendant un intervalle de temps donné.

## 2 La recherche d'objets massifs hiérarchiques dans un flux de données

### 2.1 La notion d'objet massif hiérarchique

Les flux de données que nous allons considérer ici sont de la forme $(i_t, c_t)_{t\in\mathbb{N}}$ où, pour tout instant $t \in \mathbb{N}$, l'*identifiant* $i_t$ appartient à un ensemble fini $\boldsymbol{U}$ et la *marque* $c_t$ est un nombre réel positif ou nul. Dans cet article, l'identifiant $i_t$ correspondra à une adresse IP, par exemple l'adresse destination d'un paquet IP transitant en un point $P$ donné d'un réseau, l'ensemble fini $\boldsymbol{U}$ correspondra à l'ensemble des adresses IP v4 (autrement dit chaque adresse comportera 32 bits) et la *marque* $c_t$ correspondra au nombre d'octets transportés par le paquet

considéré. Cependant, les concepts et les algorithmes que nous allons présenter ici peuvent naturellement s'appliquer à tout flux de données de la forme précédente[1]. Classiquement, sur ce type de flux de données, on définit le *compte* $a_i(\tau)$ d'un identifiant $i \in \boldsymbol{U}$ à l'instant $\tau$ par $a_i(\tau) = \sum_{t=0}^{\tau} c_t \delta_{i,i_t}$, où $\delta_{i,i_t}$ vaut 1 si $i = i_t$ et 0 sinon. Dans l'exemple ci-dessus, le compte $a_i(\tau)$ d'une adresse IP $i$ représente le nombre total d'octets à l'instant $\tau$ qui ont été envoyés vers l'adresse $i$ et qui ont transité par le point $P$ considéré. On peut alors s'intéresser aux *objets massifs* (heavy hitters) du flux à l'instant $\tau$, c'est-à-dire, aux identifiants $i$ dont le compte $a_i(\tau)$ est supérieur ou égal à une fraction $\phi N(\tau)$ du *compte total* $N(\tau) = \sum_{t=0}^{\tau} c_t$ du flux. Dans le cas particulier où l'ensemble $\boldsymbol{U}$ des identifiants étudié peut être organisé de façon hiérarchique, il peut également être intéressant d'effectuer une recherche d'objets massifs au sein de cette hiérarchie[2]. Ainsi, dans l'exemple ci-dessus, on peut regrouper les adresses IP par préfixe[3], puis associer à un préfixe donné la somme des comptes des adresses IP commençant par ce préfixe et finalement rechercher à un instant donné tous les préfixes dont le compte est supérieur ou égal à une fraction donnée du compte total du flux. L'inconvénient de cette approche est que si un préfixe donné est un objet massif, alors tous les préfixes contenus dans ce préfixe (autrement dit tous les préfixes qui sont des ancêtres du préfixe considéré) seront aussi des objets massifs, alors qu'il est parfois souhaitable de ne plus tenir compte de la contribution de cet objet massif lorsque l'on recherche des objets massifs parmi les préfixes plus courts. C'est pourquoi la notion d'*objet massif hiérarchique* (hierarchical heavy hitters) a été introduite (Cormode et al., 2003). Les objets massifs hiérarchiques d'un flux dont les identifiants appartiennent à une hiérarchie sont définis de façon récursive : les objets massifs hiérarchiques de niveau 0 (le niveau le plus bas de la hiérarchie) sont les objets massifs du flux ; les objets massifs hiérarchiques de niveau $l > 0$ sont les sommets de la hiérarchie de niveau $l$ dont la somme des comptes des identifiants qui sont leurs descendants et qui n'appartiennent pas à des objets massifs hiérarchiques de niveau inférieur à $l$, est supérieure ou égale à $\phi N(\tau)$. Pour une présentation plus détaillée de la notion d'objet massif hiérarchique, on pourra consulter Cormode et al. (2003).

## 2.2 L'algorithme de recherche d'objet massif hiérarchique de Cormode et al.

Nous allons présenter ici brièvement l'algorithme de recherche d'objet massif hiérarchique qui a été introduit dans Cormode et al. (2003) et qui a servi de base à l'algorithme de recherche que nous avons effectivement mis en œuvre. L'algorithme de Cormode et al. qui nous intéresse ici repose sur l'utilisation, à chaque niveau $l$ de la hiérarchie, d'une structure de données appelée *sketch*, qui est mise à jour à chaque instant $t$ en fonction de $i_t$ et $c_t$, et à partir de

[1] Par exemple, si on s'intéresse aux articles fréquemment achetés sur un site marchand, les algorithmes présentés dans cet article peuvent être appliqués au flux $(i_t, c_t)_{t \in \mathbb{N}}$ formé par la référence $i_t$ d'un article dans le catalogue du site et le nombre $c_t$ d'exemplaires de cet article achetés lors d'une transaction donnée.

[2] On appelle *hiérarchie* $H$ sur un ensemble $\boldsymbol{U}$ un arbre $H$ dont les sommets sont les sous-ensembles de $\boldsymbol{U}$ et tel que la racine de $H$ est l'ensemble $\boldsymbol{U}$, les feuilles de $H$ sont les éléments de $\boldsymbol{U}$ et les enfants de chaque sommet $S$ de $H$ forment une partition de $S$.

[3] Une adresse IP v4 est constituée d'une suite de 32 bits. Le *préfixe* de longueur $k$ d'une adresse IP v4 donnée est la suite des $k$ premiers bits de cette adresse. Le regroupement des adresses IP par préfixe est naturel car l'espace d'adressage IP v4 est organisé en *réseaux* et en *sous-réseaux*. Les adresses appartenant à un réseau (resp. sous-réseau) donné possèdent toutes le même préfixe, appelé *préfixe de réseau* (resp. *préfixe de réseau étendu*). Pour plus de détails sur l'organisation hiérarchique des adresses IPv4, cf. Huitema (1995).

laquelle on peut obtenir, pour tout instant $\tau$, une estimation $\hat{a}_S(\tau)$ du compte $a_S(\tau)$ de chaque sommet $S$ de niveau $l$ de la hiérarchie[4]. Pour obtenir la liste (estimée) des objets massifs hiérarchiques à un instant $\tau$, on explore récursivement la hiérarchie en commençant par le haut de la hiérarchie (cf. Cormode et al. (2003, § 4) pour plus de détails). La liste estimée obtenue coïncide avec la liste exacte lorsque toutes les comptes estimés $\hat{a}_S(\tau)$ utilisés lors de l'exploration de la hiérarchie coïncident avec le compte exact $a_S(\tau)$. Dans Cormode et al. (2003), cette estimation est effectuée à l'aide de l'algorithme *Random Subset Sums* (RSS) introduit dans Gilbert et al. (2002). L'algorithme RSS est un algorithme de Monte Carlo qui garantit que l'on a $P\{|\hat{a}_S(\tau) - a_S(\tau)| \leqslant \epsilon N(\tau)\} \geqslant 1 - \delta$, où $\epsilon$ et $\delta$ sont des réels positifs inférieurs à 1, en utilisant un espace mémoire en $O(\frac{1}{\epsilon^2}\ln(1/\delta))$ mots pour le stockage d'un sketch et un nombre d'opérations en $O(\frac{1}{\epsilon^2}\ln(1/\delta))$ pour la mise à jour d'un sketch, ainsi que pour le calcul de l'estimateur du compte d'un sommet. Par conséquent, l'algorithme que l'on obtient finalement nécessite un espace mémoire en $O(\frac{h}{\epsilon^2}\ln(1/\delta))$ mots (où $h$ est la hauteur de la hiérarchie) pour le stockage de l'ensemble des sketches, un nombre d'opérations en $O(\frac{h}{\epsilon^2}\ln(1/\delta))$ pour la mise à jour de l'ensemble des sketches et un nombre d'opérations en $O(\frac{hq}{\phi\epsilon^2}\ln(1/\delta))$ (où $q$ est le nombre maximum d'enfants de chaque sommet de la hiérarchie) pour la recherche des objets massifs hiérarchiques.

## 2.3 L'algorithme que nous avons mis en œuvre

Récemment, Cormode et Muthukrishnan (2005) ont proposé un autre type de sketch, le *Count-Min Sketch* (CMS). Comme l'algorithme RSS, l'algorithme CMS permet d'obtenir une estimation $\hat{a}_S(\tau)$ du compte $a_S(\tau)$ d'un sommet $S$ à l'instant $\tau$. Cependant, l'algorithme CMS a pour avantage de garantir que l'on a $\hat{a}_S(\tau) \geqslant a_S(\tau)$ et $P\{\hat{a}_S(\tau) \leqslant a_S(\tau) + \epsilon N(\tau)\} \geqslant 1 - \delta$ en utilisant un espace mémoire en seulement $O(\frac{1}{\epsilon}\ln(1/\delta))$ mots pour le stockage d'un sketch et un nombre d'opérations en seulement $O(\ln(1/\delta))$ pour la mise à jour d'un sketch, ainsi que pour le calcul de l'estimateur du compte d'un sommet. C'est pourquoi, comme suggéré par Cormode et Muthukrishnan (2005), nous avons utilisé l'algorithme CMS à la place de l'algorithme RSS pour estimer le compte de chaque sommet. Nous obtenons ainsi un algorithme de recherche des objets massifs hiérarchiques qui nécessite un espace mémoire en $O(\frac{h}{\epsilon}\ln(1/\delta))$ mots pour le stockage de l'ensemble des sketches, un nombre d'opérations en $O(h\ln(1/\delta))$ pour la mise à jour de l'ensemble des sketches et un nombre d'opérations en $O(\frac{hq}{\phi}\ln(1/\delta))$ (où $q$ est le nombre maximum d'enfants de chaque sommet de la hiérarchie) pour la recherche des objets massifs hiérarchiques. La réduction substantielle du nombre d'opérations nécessaire pour chaque mise à jour[5] est particulièrement appréciable. En effet, si l'on souhaite traiter le flux de données en temps réel, il est indispensable d'effectuer cette mise à jour à la cadence à laquelle on reçoit les éléments du flux ; par conséquent, en ayant nettement moins d'opérations à effectuer par mise à jour, on pourra, à puissance de calcul identique, traiter en temps réel des flux de données arrivant à une cadence bien plus élevée.

[4] En raison du nombre souvent élevé de sommets dans la hiérarchie, il serait en général trop coûteux en mémoire de tenir à jour un compteur par sommet de la hiérarchie afin d'obtenir le compte exact de chaque sommet : ainsi, le niveau le plus bas de la hiérarchie des adresses IPv4, qui est constitué de l'ensemble des adresses IPv4, comporte à lui seul $2^{32} \approx 4.10^9$ sommets.

[5] Par exemple, pour $\epsilon = 10^{-3}$, on gagne un facteur $\frac{1}{\epsilon^2} = 10^6$.

# 3 Application au cas d'un réseau IP

L'algorithme présenté au § 2.3 a été appliqué à des données réelles provenant d'un routeur Cisco installé sur un réseau IP de France Télécom. Pour des raisons pratiques, l'analyse a été effectuée sur les adresses IP des flots de données enregistrés par la sonde Netflow qui a été activée sur ce routeur (Sommer et Feldmann, 2002). La trace dont nous présentons ici l'analyse corrrespond à une vingtaine de minutes de trafic, 3,6 millions de paquets IP et 2 Go de volume. Notre analyse a été réalisée sur les adresses sources, toutes destinations confondues, pour un seuil $\phi = 10^{-2}$ que nous avons jugé représentatif des seuils choisis par les utilisateurs[6]. La hiérarchie utilisée pour l'analyse avait pour hauteur $h = 32$. Nous avons tracé le taux de faux négatifs[7] et le taux de faux positifs[8] en fonction de la précision $\epsilon$, pour différentes valeurs de la probabilité d'échec $\delta$. A titre d'exemple, nous avons représenté sur les figures 1 et 2 les courbes obtenues pour une probabilité d'échec $\delta = 10^{-2}$ [9]. Nous avons également étudié la quantité de mémoire utilisée par l'algorithme décrit au § 2.3 en fonction des paramètres $\epsilon$ et $\delta$ [10]. Les résultats que nous avons obtenus montrent que, si l'on choisit judicieusement les paramètres $\epsilon$ et $\delta$, l'algorithme présenté au § 2.3 permet d'obtenir, sur la trace étudiée, un taux de faux positifs et de faux négatifs négligeable tout en nécessitant une quantité de mémoire raisonnable en pratique : par exemple, en prenant $\delta = 10^{-2}$ et $\epsilon = 10^{-4}$, on obtient un taux de faux positifs et de faux négatifs inférieur à 1% en utilisant environ 2 millions de mots.

# 4 Conclusion

Comme suggéré par Cormode et Muthukrishnan (2005), nous avons utilisé l'algorithme CMS à la place de l'algorithme RSS afin de rechercher les objets massifs hiérarchiques à l'aide de la méthode présentée par Cormode et al. (2003, § 4). Cependant, alors que Cormode et Muthukrishnan (2005) ne préconisent cette solution que dans le cas de flux de données dont la marque $c_t$ peut être négative[11], nous avons délibérément appliqué cet algorithme au cas de flux de données dont la marque $c_t$ est obligatoirement positive ou nulle. En effet, les méthodes proposées par Cormode et al. (2003, § 3) pour rechercher les objets massifs hiérarchiques dans des flux de données dont la marque $c_t$ est obligatoirement positive ou nulle présentent pour nous les deux inconvénients suivants : d'une part ces méthodes nécessitent une quantité de mémoire qui (pour une précision donnée) augmente avec le compte total $N(\tau)$[12] et d'autre part

[6] En effet, le nombre maximum d'objets massifs hiérarchiques à un instant $\tau$ donné est majoré par $1/\phi$ (Cormode et al., 2003). En prenant un seuil $\phi = 10^{-2}$, on obtient donc tout au plus une centaine d'objets massifs hiérarchiques, ce qui convient souvent aux utilisateurs.

[7] On appelle *faux négatif* un objet massif hiérarchique qui n'a pas été détecté par l'algorithme.

[8] On appelle *faux positif* un préfixe qui a été considéré à tort par l'algorithme comme étant un objet massif hiérarchique.

[9] En réduisant d'avantage la valeur de $\delta$, nous n'avons obtenu qu'une légère amélioration des performances alors que la quantité de mémoire utilisée augmentait considèrablement.

[10] Nous ne présentons pas ces courbes ici par manque de place.

[11] Cf. Cormode et al. (2003) pour un exemple de flux de données dont la marque $c_t$ peut être négative.

[12] Pour notre application, cette dépendance en $N(\tau)$ est très gênante car elle ne nous permet pas de choisir *a priori* la précision de l'algorithme qui permettra d'utiliser au mieux la mémoire disponible. En effet, avec ces méthodes, si on choisit une précision élevée, qui exigera donc beaucoup de mémoire et qui sera ainsi adaptée à un faible trafic, on risquera de manquer de mémoire si le trafic est plus élevé que prévu durant la période étudiée ; *a contrario*, si on choisit une précision faible, qui exigera donc peu de mémoire et qui sera ainsi adaptée à un trafic élevé, on aura

ces méthodes ont été conçues uniquement pour rechercher les objets massifs hiérarchiques dans des flux de données dont la marque $c_t$ est obligatoirement positive ou nulle ; leur adaptation au cas de flux de données dont la marque $c_t$ peut être négative nous paraît très difficile. La solution que nous avons mise en œuvre ne présente pas ces inconvénients. Dans notre cas, le nombre de mots nécessaire en mémoire reste constant avec le temps et en particulier ne dépend pas du compte total $N(\tau)$ : il n'est pas nécessaire d'estimer finement à l'avance le volume de trafic qui sera observé ni la durée totale pendant laquelle l'analyse sera effectuée[13]. De plus, comme la mémoire nécessaire aux méthodes de Cormode et al. (2003, § 3) a pour majorant $O(\frac{h}{\epsilon}\ln[\epsilon N(\tau)])$ et que la solution que nous avons mise en œuvre nécessite $O(\frac{h}{\epsilon}\ln(1/\delta))$ mots, on peut considérer que les deux types d'algorithmes occupent une mémoire du même ordre de grandeur lorsque l'on a $N(\tau) \approx \frac{1}{\epsilon\delta}$. Pour $\delta = 10^{-2}$ et $\epsilon = 10^{-4}$, on obtient un compte total de $N(\tau) \approx 10^6$, qui est rapidement atteint en pratique dans l'application que nous avons étudiée. Par conséquent, la solution que nous avons mise en œuvre est bien pertinente et nous permet de mieux exploiter la mémoire disponible que les méthodes proposées par Cormode et al. (2003, § 3). D'autre part, l'adaptation de la solution que nous avons mise en œuvre au cas de flux de données dont la marque $c_t$ peut être négative est extrêmement simple : il suffit pour cela de modifier l'estimation du compte de chaque préfixe en remplaçant la minimisation qui intervient dans cette estimation par une recherche de médiane (Cormode et Muthukrishnan, 2005). La solution que nous avons mise en œuvre possède donc une grande flexibilité qui nous permettra de réutiliser plus facilement les programmes déjà écrits lorsque nous étudierons des flux de données dont la marque $c_t$ peut être négative.

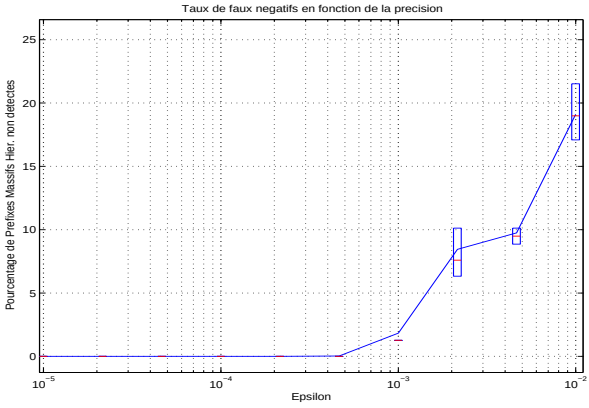


**FIG. 1** – *Taux de faux négatifs en fonction de la précision $\epsilon$. La probabilité d'échec était fixée à $\delta = 10^{-2}$. Pour chaque valeur de la précision $\epsilon$, l'algorithme présenté au § 2.3 a été exécuté 40 fois. On a représenté sur cette figure le taux moyen de faux négatifs ainsi que les boxplots correspondant au 1$^{er}$quartile, à la médiane et au 3$^{e}$ quartile.*

sous-utilisé la mémoire disponible si le trafic est plus faible que prévu durant la période étudiée.

[13] Il faut simplement s'assurer au préalable que la taille des mots utilisée reste suffisante pour toutes les situations envisageables.

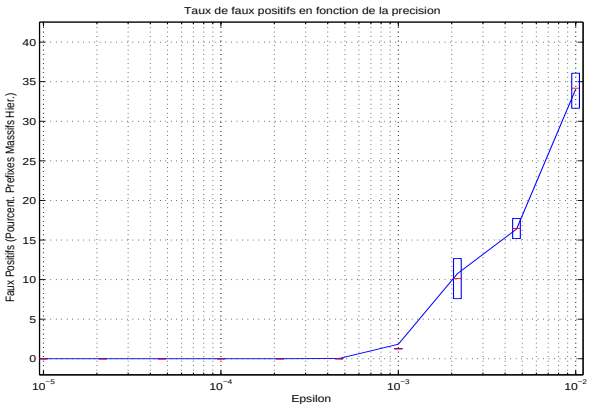


**FIG. 2** – *Taux de faux positifs en fonction de la précision $\epsilon$. La probabilité d'échec était fixée à $\delta = 10^{-2}$. Pour chaque valeur de la précision $\epsilon$, l'algorithme présenté au § 2.3 a été exécuté 40 fois. On a représenté sur cette figure le taux moyen de faux positifs ainsi que les boxplots correspondant au $1^{er}$ quartile, à la médiane et au $3^{e}$ quartile.*

Les auteurs remercient Guillaume Picard pour ses commentaires très pertinents sur une version préliminaire de cet article.

## Références

Cormode, G., F. Korn, S. Muthukrishnan, et D. Srivastava (2003). Finding hierarchical heavy hitters in data streams. In *International Conference on Very Large Databases*, pp. 464–475.

Cormode, G. et S. Muthukrishnan (2005). An improved data stream summary : The count-min sketch and its applications. *Journal of Algorithms*.

Gilbert, A. C., Y. Kotidis, S. Muthukrishnan, et M. Strauss (2002). How to summarize the universe : Dynamic maintenance of quantiles. In *VLDB*, pp. 454–465.

Huitema, C. (1995). *Le routage dans l'Internet*. Editions Eyrolles.

Sommer, R. et A. Feldmann (2002). Netflow : information loss or win ? In *IMW '02 : Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, New York, NY, USA, pp. 173–174. ACM Press.

## Summary

Over the past years, many stream mining algorithms have been proposed to perform real-time analysis on data streams. In this paper, we show how we have used recent stream mining algorithms for online identification of hierarchical heavy hitters to find in real-time in IP traffic data those IP prefixes whose frequencies exceed a specified threshold.