HAL open science

PULP: Un protocole epidémique hybride

Pascal Felber, Anne-Marie Kermarrec, Lorenzo Leonini, Étienne Rivière, Spyros Voulgaris

To cite this version:

Pascal Felber, Anne-Marie Kermarrec, Lorenzo Leonini, Étienne Rivière, Spyros Voulgaris. PULP: Un protocole epidémique hybride. AlgoTel, 2009, Carry-Le-Rouet, France. inria-00383063

HAL Id: inria-00383063

https://inria.hal.science/inria-00383063v1

Submitted on 11 May 2009

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

PULP: Un protocole épidémique hybride

Pascal Felber[1], Anne-Marie Kermarrec[2], Lorenzo Leonini[1],
Étienne Rivière[3] et Spyros Voulgaris[4]

[1] *Institut d'Informatique, Université de Neuchâtel, Suisse.* [2] *INRIA Rennes-Bretagne Atlantique, France.*
[3] *NTNU Trondheim, Norvège.* [4] *Vrije Universiteit Amsterdam, Pays-Bas.*

Les protocoles épidémiques offrent une solution simple et robuste pour la diffusion d'informations dans les systèmes répartis à grande échelle. Ils utilisent des communications périodiques entre couples de pairs, choisis de manière aléatoire. Leur efficacité et leur robustesse résultent de la multiplicité des échanges et de communications redondantes. Néanmoins, les algorithmes proposés jusqu'à présent tiennent peu compte de la charge sur le réseau, induite par les communications redondantes et par les opérations périodiques, et des compromis implicites entre cette charge et l'efficacité de la diffusion. Cet article décrit PULP, un protocole qui combine l'efficacité des opérations de type *fournir* (push) et la légèreté de communications de type *demander* (pull) dont la fréquence s'adapte au rythme de publication.

Mots clés: Diffusion robuste d'information, Protocoles épidémiques

1 Introduction

Les protocoles épidémiques sont totalement décentralisés et fondés sur des interactions à intervalle régulier entre les nœuds d'un réseau. Ils sont particulièrement adaptés à la grande échelle et au dynamisme, et fournissent des solutions robustes et de conception simple pour diverses tâches : construction de réseaux logiques, agrégation de valeurs, etc. L'application la plus commune est la diffusion d'informations [1, 3, 4] (mises à jour, observation, etc.). L'élément d'information échangé se propage de proche en proche à la façon d'une épidémie dans un groupe humain. On distingue deux types de transfert d'informations lors d'une diffusion épidémique : par des opérations de type *fournir* (push), et par des opérations de type *demander* (pull) périodiques. Le premier type, *fournir*, est initié lorsqu'un nœud reçoit une information *pour la première fois*, ou lorsque ce nœud crée une nouvelle information. La donnée est transmise immédiatement à f (*fanout*, ou facteur de dispersion) autres nœuds, en décrémentant un *temps de vie* (TTL) associé au message. Lorsque ce temps de vie est nul, le message n'est plus retransmis. Le second type, dit *demander* (pull) est lui fondé sur les échanges périodiques : à chaque période, chaque nœud contacte un autre nœud choisit aléatoirement en indiquant les identifiants de messages déjà reçus ; il reçoit en retour ceux dont il n'avait pas connaissance. Dans les deux cas, les pairs aléatoires sont obtenus soit en connaissant l'intégralité des pairs participants [2], ce qui ne passe pas à l'échelle, ou en utilisant un service décentralisé d'échantillonnage de pairs, lui-même construit à l'aide d'un protocole épidémique [3]. Un tel service d'échantillonnage maintient sur chaque nœud une liste de c pairs, en assurant le retrait des nœuds fautifs ou supprimés et l'ajout des nouveaux nœuds, par exemple en utilisant un processus de brassage de liens du protocole Cyclon [8]. On appelle *ronde* une période d'exécution du protocole épidémique sur tous les nœuds, par ex. l'initiation d'une *demande* ou d'un brassage de liens dans Cyclon. Cette notion sert à l'analyse mais aucune synchronisation n'est bien sûr nécessaire entre les pairs en pratique. Bien que les opérations *fournir* puissent en pratique être exécutées dès la réception du message, pour les besoins de l'explication on appellera aussi *ronde* l'envoi de messages avec un TTL t par tous les pairs notifiés pour la première fois par un message de TTL $t+1$.

Une diffusion de type *fournir* notifie l'ensemble des nœuds d'un réseau en $O(\log n)$ rondes avec forte probabilité si $f = O(\log n)$, où n est la taille du réseau. Néanmoins, cette diffusion s'accompagne d'un grand nombre de réceptions multiples, ou *doublons*. Le nombre de nœuds notifiés augmente de manière exponentielle pendant les premières rondes, puis après que la moitié du réseau a été notifiée, le nombre de nœuds non notifiés se réduit d'un facteur constant par ronde [1]. Avant cela, le nombre de doublons augmente significativement. La figure 1 présente l'évolution du nombre de pairs notifiés, du nombre de messages « utiles » (première réception) et inutiles (doublons) pour $f = 2$ et $f = 4$, avec un TTL infini,

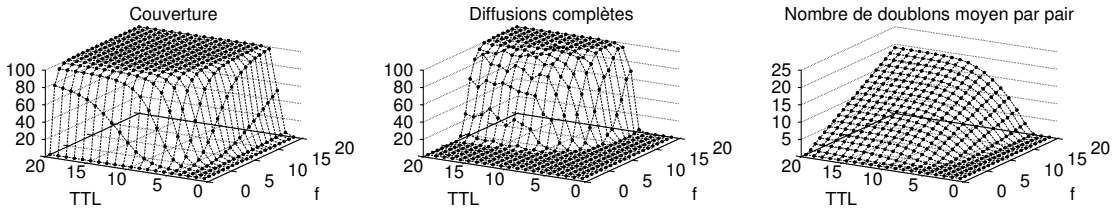


FIG. 2: Performance de la diffusion de type *fournir* : évolution de la couverture moyenne pour chaque message, de la couverture atomique (proportion des messages reçus par *tous* les nœuds), et taux de doublons associé.

dans un réseau de 10000 nœuds. On observe qu'un nombre de messages faible (tant utiles que de doublons) ne permet pas de toucher l'ensemble du réseau, mais que trois doublons par nœud sont nécessaires pour obtenir une couverture de 98.09%. La figure 2 présente la performance de 200 diffusions dans un réseau de 10000 nœuds : la couverture moyenne pour chaque diffusion, le taux de diffusions atomiques (reçues par tous), et le taux de doublons. On observe que si, pour un message, un TTL et un facteur de dispersion f faibles permettent d'obtenir une couverture presque complète en moyenne, il est nécessaire d'utiliser de grandes valeurs pour des couvertures réellement complètes, ce qui entraîne énormément de doublons.

Les diffusions par *demandes* entraînent un coût permanent sur le réseau, de par les requêtes périodiques entre pairs. Celles-ci sont de petite taille, mais n requêtes sont émises à chaque ronde. Le délai de réception dépend de leur fréquence, mais il est inefficace d'engendrer un grand nombre de requêtes en permanence alors qu'elles sont efficaces lors des diffusions seulement. De plus, comme montré par la figure 1, la performance de la diffusion pendant les premières rondes est faible ; comme chaque pair est sélectionné en moyenne une fois par ronde, en moyenne aussi un seul nouveau pair est notifié à la première ronde, deux à la suivante, etc. La diffusion par *demandes* est donc plus efficace une fois qu'un nombre conséquent de nœuds ont connaissance de l'information.

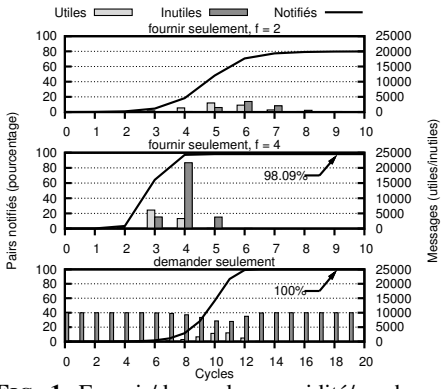


FIG. 1: Fournir/demander : rapidité/surcharge (simulation, topologie en graphe aléatoire).

Enfin, les protocoles communément utilisés [1, 3, 4] considèrent des messages indépendants ; il est intéressant de prendre en compte le cas où des ensembles de messages sont envoyés, afin d'adapter la réactivité de la diffusion à leur fréquence, ainsi que pour utiliser les messages transmis pour indiquer aux pairs contactés la présence d'autres messages en cours de transmission.

Le protocole décrit par cet article, PULP, combine les bonnes propriétés de ces deux types de protocoles, et ajoute plusieurs améliorations pour pallier certaines de leurs limitations, comme le compromis implicite et fixé à l'avance entre fréquence des demandes et performance de la diffusion. PULP considère des flux, réguliers ou non, de messages émis par des nœuds quelconques du réseau. Il combine une phase de diffusion de type *fournir* limitée, évitant les doublons, à des opérations périodiques de *demandes* dont la fréquence s'adapte au rythme de publication.

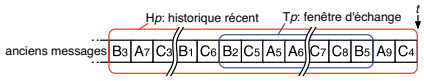
Le protocole *Interleave* [7] combine lui aussi des opérations *fournir* et *demander* mais est fondé sur un modèle différent de celui de PULP, où l'ensemble des messages sont ordonnés et issu d'un seul et même nœud du réseau (tandis que PULP considère des écrivains multiples et non synchronisés). Interleave ne considère pas de fréquence des demandes adaptatives comme le fait PULP, mais pourrait en être doté.

2 Le protocole PULP

L'algorithme 1 présente les opérations de PULP pour un nœud P, dont nous décrivons ici les composants. **Phase *fournir* limitée.** Afin d'éviter les doublons de réception des messages tout en conservant la phase de croissance rapide pour initier la diffusion, la portée de la diffusion par opérations *fournir* est limitée, p.ex. à 5% du réseau. À cet effet, les valeurs de TTL et de f sont initialement décidées pour couvrir ces 5%. La taille du réseau n'est pas *a priori* connue des nœuds : il faut utiliser une estimation. Un exemple d'algorithme [5] augmente les messages du protocole de maintenance du réseau Cyclon [8], utilisé par notre

Algorithme 1 : Algorithme PULP sur le nœud P

Variables ($\Delta_{ajuster}$, *TTL–temps de vie–* et f sont des constantes du protocole)
- H_P : historique des identifiants des messages reçus (récemment)
- $\Delta_{demander}$: période des requêtes « demander » (initialisée à 30 secondes)
- *manquant* : ensemble des identifiants de messages connus, mais pas encore reçus
- *tailleManquantPrécédente* : taille de *manquant* à la fin de la dernière période $\Delta_{ajuster}$
- $préc_{utile}$: nombre de réponses non vides (utiles) à une requête « demander » durant la période $\Delta_{ajuster}$ courante
- $préc_{inutile}$: nombre de réponses vides (inutiles) à une requête « demander » durant la période $\Delta_{ajuster}$ courante




```
/* Message envoyé sans demande préalable à P par Q                    */
méthode FOURNIR(message, sauts, Q, T_Q)
    /* Poursuivre la diffusion si nécessaire                          */
    si message est reçu pour la première fois alors
        ajoute message à H_P
        si sauts > 0 alors
            appeler FOURNIR(message, sauts-1, P, T_P) sur f nœuds
            choisis aléatoirement
    /* Les msgs. seront demandés à la prochaine op. demander          */
    manquant ← manquant \ {message} ∪ {m ∈ T_Q : m ∉ H_P}

/* Demande périodique des éléments manquants                          */
processus REQUÊTE « DEMANDER » PÉRIODIQUE()
    faire toutes les Δ_demander secondes
        /* Le mélange réduit la prob. de réception de doublons        */
        mélange manquant
        appeler DEMANDE(manquant, P, T_P) sur un nœud aléatoire Q

/* Appelé à distance par un nœud Q qui demande un message à P         */
méthode DEMANDE(demandé, Q, T_Q)
    m ← premier élément de demandé ∈ T_P, ou ⊥ si aucun de tel
    appeler RÉPONDRE À DEMANDE(m, P, T_P) sur Q

/* Réponse à une requête demander de la part de P                     */
méthode RÉPONDRE À DEMANDE(message, Q, T_Q)
    si message = ⊥ ∨ message ∈ H_P alors
        préc_inutile ← préc_inutile + 1
    sinon
        ajoute message à H_P
        manquant ← manquant \ {message} ∪ {m ∈ T_Q : m ∉ H_P}
        préc_utile ← préc_utile + 1

/* Ajustement périodique de la période de demandes sur le nœud P      */
processus ADAPTEPÉRIODE()
    faire toutes les Δ_ajuster secondes
        si |manquant| > tailleManquantPrécédente alors
            Δ_demander ← Δ_ajuster / (|manquant| − tailleManquantPrécédente + préc_utile)
        sinon
            si |manquant| > 0 ∧ préc_inutile ≤ préc_utile alors
                Δ_demander ← Δ_demander × 0.9
            sinon
                Δ_demander ← Δ_demander × 1.1
        Δ_demander ← max(Δ_demander, Δ_demander_min)
        Δ_demander ← min(Δ_demander, Δ_demander_max)
        préc_inutile ← 0, préc_utile ← 0
        tailleManquantPrécédente ← |manquant|
```

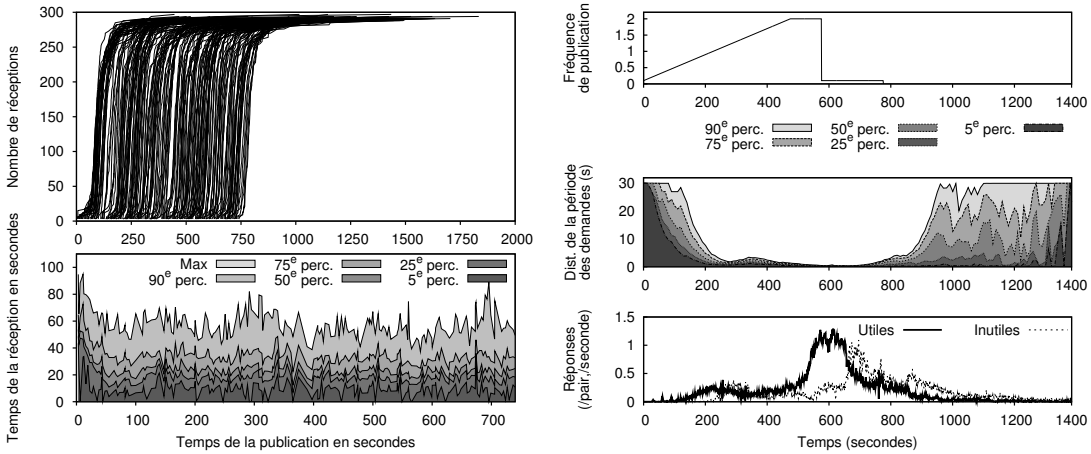
mise en œuvre, par des échantillonnages de hachage des identifiants des nœuds, et utilise la densité de ces échantillons dans une plage de valeurs pour obtenir la taille du réseau : pour 1000 nœuds, l'utilisation de TTL=2 et $f = 6$ permet de diffuser initialement à 4% du réseau, et entraîne en moyenne 0.1% de doublons.

Fenêtre d'échange. Chaque pair conserve un historique des messages reçus H_P, qui peut être de taille limitée (voir figure en haut à droite de l'algorithme 1), et inclut avec ses messages (*fournir* et *demander*) une fenêtre d'échange des messages récents T_P. Celle-ci permet de notifier les pairs de l'existence de nouveaux messages, ce qui permet (1) d'adapter la fréquence des requêtes *demander*, et (2) de demander les messages aux pairs qui en ont indiqué l'existence (omis dans l'algorithme 1). Notons que si la phase *fournir* initiale pour un message ne touche chaque nœud qu'avec une probabilité p, une suite de messages touchera à chaque fois un ensemble différent de nœuds et la probabilité pour un nœud de ne pas être touché par m messages sera donc $(1-p)^m$. Ainsi, si $p = 50\%$, les seules phases *fournir* de 14 messages notifient 52% du réseau de l'existence des publications ; ce qui s'ajoute aux notifications obtenues lors des *demandes*, où T_P est aussi envoyé. La fenêtre d'échange ne comprend pas les plus récents messages afin d'éviter les doublons de réception à la fois par *fournir* et *demander*. La taille de la fenêtre d'échange peut varier en fonction de la fréquence de publication (p.ex. les messages restent dans T_P pour au moins s cycles).

Phase *demander* à fréquence adaptative. Les messages dont l'existence est connue mais qui n'ont pas encore été reçus sont inclus dans l'ensemble *manquant*, et seront explicitement *demandés* par les requêtes périodiques. La période variable d'exécution du processus de requêtes *demander* est Δ_{demander}. Elle est ajustée par un processus indépendant ADAPTEPÉRIODE, exécuté toutes les Δ_{ajuster} secondes. Celui-ci adapte Δ_{demander} dans la plage autorisée en fonction de l'utilité des réponses aux demandes pendant la période précédente, et de la taille de l'ensemble *manquant* : si la taille de ce dernier augmente, la fréquence de *demande* augmente elle aussi et s'ajuste à la fréquence nécessaire pour suivre le rythme de publication ; sinon, Δ_{demander} est adaptée de manière à équilibrer les réponses utiles et inutiles.

3 Évaluation du protocole sur PlanetLab

PULP a été déployé sur 300 nœuds de la plate-forme PlanetLab (`http://www.planet-lab.org`) à l'aide du langage et des outils fournis par SPLAY [6]. Chaque machine PlanetLab supporte un nœud PULP. La figure 3(a) présente les délais de réception de 300 messages, soumis à la fréquence d'un toutes les 3.7 secondes à partir de nœuds aléatoires : cumul du nombre de réception pour chaque message en haut



(a) Réception de 300 messages (1 toutes les 3.7 secondes)

(b) Évolution de Δ_{demander}

FIG. 3: Performances sur 300 nœuds PlanetLab.

et évolution de la fonction de répartition en bas. Les machines sont particulièrement chargées, néanmoins l'intégralité des nœuds reçoit tous les messages, ce qui serait difficile à atteindre avec une simple diffusion en arbre ou au dessus d'une table de hachage répartie, par exemple. En moins de 30 secondes après sa création, chaque message est reçu par la moitié du réseau et les derniers nœuds (les plus chargés) le reçoivent en une minute environ. Ceci est à mettre en rapport avec l'extrême légèreté du protocole, chaque nœud n'envoyant que 16 à 17 messages UDP par minute en moyenne. On observe également la période de mise en route lors de la diffusion par *fournir* des premiers messages, qui initient l'augmentation de la fréquence des *demandes* des autres nœuds par la suite, et l'adaptation à la fréquence de publication. La figure 3(b) illustre cette capacité d'adaptation de la période Δ_{demander} à la fréquence de publication. Ici, un seul nœud publie avec une fréquence variable (en haut) : d'un message toutes les 5 secondes à 2 messages par seconde, suivi d'une chute brusque. On observe (au milieu) la distribution de Δ_{demander} sur les 300 nœuds et (en bas) le nombre de réponses à ces *demandes* selon leur utilité. La période Δ_{demander} est initialisée à 30 secondes et décroît rapidement avec l'augmentation de la fréquence de publication. Elle reste faible pour terminer la diffusion des messages en cours après la chute de la fréquence de publication, puis revient à de grandes valeurs une fois qu'il n'existe plus de messages en *transit* au sein du réseau. Cette évolution est fonction du nombre de *demandes* utiles et inutiles : l'apparition de demandes inutiles entraîne une augmentation de Δ_{demander}, et le nombre de messages non encore reçus sa diminution. L'adaptation de la fréquence permet d'obtenir une performance correcte lorsqu'une activité importante existe sur le réseau et de limiter le coût résiduel lorsqu'il n'y en a pas. En outre, l'utilisation de la phase initiale *fournir* permet (1) d'initialiser la diffusion rapidement et (2) de propager des informations sur les messages en transit. Ces aspects permettent de pallier les principales limitations des deux types de protocoles utilisés séparément : délais, taux de doublons, compromis entre coût lors des phases d'inactivité et délais lors des phases d'activité pour les *demandes* périodiques.

Références

[1] K.P. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, et Y. Minsky. Bimodal multicast. *ACM TOCS*, 17(2), 1999.

[2] A.J. Demers, D.H. Greene, C. Hauser, W. Irish, et J. Larson. Epidemic algorithms for replicated database maintenance. In *Proceedings PODC'87*, pages 1–12, Vancouver, Canada, août 1987.

[3] Mark Jelasity, Spyros Voulgaris, Rachid Guerraoui, Anne-Marie Kermarrec, et Maarten van Steen. Gossip-based peer sampling. *ACM Transactions on Computer Systems*, 25(3), août 2007.

[4] R. Karp, C. Schindelhauer, S. Shenker, et B. Vocking. Randomized rumour spreading. In *IEEE Proc. 41st Ann. Symp. Foundations of Computer Science (FOCS)*, 2000.

[5] D. Kostoulas, D. Psaltoulis, I. Gupta, K. P. Birman, et A. J. Demers. Active and passive techniques for group size estimation in large-scale and dynamic distributed systems. *J. of Systems and Software*, 80(10) :1639–1658, 2007.

[6] L. Leonini, E. Rivière, et P. Felber. SPLAY : Distributed systems evaluation made simple (or how to turn ideas into live systems in a breeze). In *Proceedings of NSDI'09*, Boston, MA, USA, avril 2009.

[7] Sujay Sanghavi, Bruce Hajek, et Laurent Massoulié. Gossiping with multiple messages. In *INFOCOM*, pages 2135–2143, 2007.

[8] S. Voulgaris, D. Gavidia, et M. van Steen. CYCLON : Inexpensive membership management for unstructured P2P overlays. *J. of Network and Systems Management*, 13(2), juin 2005.