RFC 6206 : The Trickle Algorithm

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 29 mars 2011. Dernière mise à jour le 30 mars 2011

Date de publication du RFC : Mars 2011

https://www.bortzmeyer.org/6206.html

Dans le cadre des travaux sur l'Internet des objets, le groupe de travail ROLL <http://tools.ietf.org/wg/roll> de l'IETF s'occupe de définir des mécanismes de routage pour des réseaux d'engins aux capacités limitées, et connectés par des liens radio à faible portée et souvent brouillés. (Par exemple, cela peut décrire la situation de capteurs dans une usine.) Une brique de base pour beaucoup de protocoles utilisés dans ce contexte est un algorithme pour coordonner un état entre toutes les machines (cet état pouvant être par exemple la table de routage, mais aussi une configuration commune). Trickle (terme désignant un mince filet d'eau, par exemple celui coulant du robinet en cas de fuite) est l'algorithme utilisé et ce RFC est sa première normalisation. Il est notamment utilisé dans le protocole RPL du RFC 6550 [1].

Vu le contexte très spécial, les protocoles classiques de distribution d'information ne convenaient pas forcément à ce type de réseau, où il est crucial d'économiser l'énergie, et de fonctionner malgré des liaisons physiques sommaires. Trickle atteint ces objectifs en ne diffusant l'information que lorsque elle a changé. L'idée de base est de permettre à deux nœuds de déterminer très rapidement s'ils ont la même version de l'information distribuée et, sinon, de se synchroniser. S'ils sont synchronisés, il n'y a plus beaucoup de communication. Lorsque de la nouvelle information apparait, le trafic reprend. Trickle repose entièrement sur les mécanismes de diffusion du réseau local et n'est donc pas routé. Le tout est simple et tient dans 50 à 200 lignes de code C (je n'ai pas trouvé de distribution officielle de référence de ce code miracle mais l'implémentation dans Contiki fait effectivement cette taille).

Trickle est un algorithme générique : conçu à l'origine pour distribuer du code (pour mettre à jour le programme sur les autres machines), il a pu être adapté pour distribuer n'importe quel type d'information, par exemple une table de routage. Pour un nouveau type d'information, il suffit de définir ce que veut dire « être synchrone » ou pas et la partie générique de Trickle fait le reste.

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc6550.txt

Assez de publicité, place à l'algorithme (sections 3 e 4 du RFC). Le travail de chaque nœud est de transmettre les données qu'il connait jusqu'à ce que, en entendant les autres nœuds, il se rende compte que ses transmissions sont inutiles (cela veut dire qu'un nœud Trickle isolé continuera à parler tout seul). Pour prendre l'exemple du protocole CTP ("*Collection Tree Protocol*"), quelques paquets par heure suffisent.

La transmission se fait à une adresse "*multicast*" locale. Par exemple, au dessus d'IPv6, Trickle va écrire à une adresse "*multicast*" locale au lien alors qu'en IPv4, il utiliser `255.255.255.255`.

Lorsqu'un nœud reçoit les messages d'un autre, il y a deux cas :

— Le message montre que celui dont on vient de recevoir des nouvelles a la même version de l'information,
— Le message montre qu'un des deux nœuds, le récepteur ou l'émetteur, est en retard. Notons que Trickle ne fait pas trop de différence entre ces deux sous-cas : ils entraineront tous les deux une transmission de données.

L'un des principaux paramètres d'un algorithme spécifique utilisant Trickle est un moyen de déterminer cette notion de « être à jour ». Par exemple, supposons que l'information ait un numéro (comme le numéro de série des zones DNS). Alors, « être à jour » signifie avoir le même numéro que l'autre et « être en retard » avoir un numéro inférieur.

Pour décider ensuite de l'action exacte à entreprendre, les nœuds Trickle ont besoin de quelques paramètres :

— la taille minimum de l'intervalle, Imin,
— la taille maximum de l'intervalle, Imax,
— la constante de redondance k.

et de quelques variables :

— la taille actuelle de l'intervalle, I,
— le moment de la transmission, t,
— le compteur c.

Le nœud suit les règles suivantes :

— Choisir I entre Imin et Imax. Commencer le premier intervalle.
— Au début de l'intervalle, mettre c à zéro et t à une valeur située dans la deuxième moitié de l'intervalle.
— À chaque message reçu pour lequel on est à jour, incrémenter c.
— Au temps t, si c est inférieur à k, on transmet les données. (C'est la seule étape de l'algorithme où on transmet quelque chose.) c ¡ k peut se traduire par « j'ai trop de voisins qui ne sont pas à jour par rapport à l'ensemble de mes voisins ». C'est par cette comparaison de c (nombre de voisins à jour) à k que Trickle s'adapte automatiquement à la densité du réseau (s'il y a plein de voisins à jour, pas la peine de se fatiguer à transmettre, un autre le fera).
— À la fin de l'intervalle, doubler la taille de l'intervalle (en s'arrêtant à Imax).
— Si on entend un message qui montre que quelqu'un (le nœud qui exécute cet algorithme ou bien son voisin) n'est pas à jour, on réduit l'intervalle à Imin. L'idée est de rendre l'algorithme plus dynamique lorsqu'il y a des nœuds qui sont en retard.

On note que Trickle ne réagit pas immédiatement en détectant un retard : il se contente de démarrer un nouvel intervalle. C'est fait exprès pour éviter une tempête de mises à jour, par exemple lorsqu'un nouveau nœud rejoint le réseau après une absence. Trickle privilégie donc l'économie de ressources, et ne cherche pas à être temps-réel.

Ça, c'était l'algorithme. Simple, non ? Maintenant, si vous êtes concepteur de protocoles et que vous voulez utiliser Trickle, que vous reste-t-il à faire ? La section 5 liste vos tâches :

— Préciser des valeurs par défaut pour les paramètres Imin, Imax, k, etc. Notamment, Imin dépend de la latence du réseau utilisé.
— Définir ce que signifie « être à jour » et « être en retard ».

— Et quelques détails. Regardez le RFC 6550 pour un exemple de protocole de routage utilisant Trickle.

Il ne faut quand même pas oublier quelques considérations pratiques (section 6). D'abord, le bon fonctionnement de Trickle dépend d'un accord de tous les nœuds sur les paramètres à utiliser. En cas de différence, le résultat peut être sous-optimal voire faux dans certains cas. Par exemple, un nœud qui aurait une constante de redondance k supérieure aux autres pourrait se retrouver à émettre à chaque intervalle, vidant sa batterie. De même, un désaccord sur Imax pourrait mener certains nœuds à ne jamais transmettre, laissant les autres faire tout le travail. En outre, toutes les valeurs ne sont pas forcément raisonnables. Par exemple, la constante de redondance k mise à l'infini est théoriquement possible (Trickle travaillerait alors en mode bavard, n'économisant pas les transmissions) mais très déconseillée. Il faut donc trouver un moyen de distribuer la bonne configuration à toutes les machines.

Le pire problème serait évidemment une fonction de définition de la cohérence différente selon les nœuds, cela pourrait mener à une impossibilité de converger vers un état commun. C'est pour cela que le RFC demande qu'elle soit strictement définie pour un protocole donné, et non configurable.

L'auteur de protocole, ou le programmeur, pourraient vouloir « améliorer » Trickle en ajustant l'algorithme. Attention, avertit le RFC, cet algorithme a été soigneusement étudié et testé et la probabilité d'arriver à l'améliorer est faible. Il est conseillé de ne pas le toucher gratuitement. Trickle est extrêmement simple et la plupart des améliorations mèneraient à un code plus compliqué. Or, pour le genre de machines pour lesquelles Trickle est prévu, économiser en transmissions de paquets pour finir par faire davantage de calculs ne serait pas forcément optimal.

Enfin, question sécurité, c'est simple, Trickle n'en a pas (section 9). Un méchant peut facilement pousser tous les nœuds à transmettre, épuisant leurs ressources. Et il peut assez facilement empêcher le groupe de converger vers un état stable. Les protocoles qui utilisent Trickle doivent donc définir leurs propres mécanismes de sécurité.

Pour approfondir Trickle, notamment sur ses performances ou son implémentation, les articles recommandés sont « "*Trickle : a self-regulating algorithm for code propagation and maintenance in wireless sensor networks*" <http://stanford.edu/~neilp/pubs/trickle-nsdi04.pdf> » et « "*The emergence of a networking primitive in wireless sensor networks*" <http://portal.acm.org/citation.cfm?id=1364804> ».

Plusieurs utilisations de Trickle sont mentionnées en section 6.8, avec références. Trickle est par exemple utilisé dans Contiki : <http://www.sics.se/~adam/contiki/contiki-2.0-doc/a00543.html>. Damien Wyart s'est attelé à la tâche de trouver d'autres mises en œuvre et voici ses résultats (je le cite).

« Dans les travaux qui parlent de Trickle, on voit souvent cité Maté <http://www.eecs.berkeley.edu/~pal/mate-web/>, une sorte d'environnement qui permet de mettre en réseau des mini-machines virtuelles (si j'ai bien compris), et certains documents indiquent que Maté contient une implantation de Trickle. Maté est distribué principalement sous forme de RPM, ce qui n'aide pas sur Debian... Il ne semble plus évoluer depuis 2005 (mais l'auteur principal <http://csl.stanford.edu/~pal/> continue à travailler sur TinyOS et peut sans doute être contacté pour obtenir plus de détails sur les implantations de Trickle. De plus, Maté est intimement lié à TinyOS qui a depuis, semble-t-il, beaucoup évolué. »

« Par recoupements (c'est assez rare d'avoir à autant jouer les détectives entre fichiers RPM, confusion sur les versions et vieux dépôts CVS — TinyOS a maintenant un SVN chez Google Code), je suis arrivé principalement au fichier <http://tinyos.cvs.sourceforge.net/viewvc/tinyos/tinyos-1.x/tos/lib/VM/components/MVirus.nc?view=log>. Sur une copie locale des deux sous-arborescences `tinyos-1.x/tools/java/net/tinyos/script` et `tinyos-1.x/tos/lib/VM` (citées dans <http://www.eecs.berkeley.edu/~pal/mate-web/downloads.html>), j'arrive à ceci (avec le bien pratique concurrent de grep, ack <http://betterthangrep.com/>) :

```
dw@brouette:/storage/sandbox/tinyos-1.x$ ack -ai trickle
tos/lib/VM/components/MVirus.nc
118:  MateTrickleTimer versionTimer;
119:  MateTrickleTimer capsuleTimer;
153:  void newCounter(MateTrickleTimer* timer) {

tos/lib/VM/doc/tex/mate-manual.tex
615:that has newer code will broadcast it to local neighbors. The Trickle
635:Trickle's suppression operates on each type of packet (version,
645:\subsubsection{Trickle: The Code Propagation Algorithm}
647:The Trickle algoithm uses broadcast-based suppressions to quickly
651:completes, Trickle doubles the size of an interval, up to
662:Trickle maintains a redundancy constant $k$ and a counter

tos/lib/VM/types/Mate.h
168:typedef struct MateTrickleTimer {
173:} MateTrickleTimer;

tools/java/net/tinyos/script/VMFileGenerator.java
524:    writer.println(" * MVirus uses the Trickle algorithm for code propagation and maintenance.");
528:    writer.println(" * \"Trickle: A Self-Regulating Algorithm for Code Propagation and Maintenance");
549:    writer.println("     the timer interval for Trickle suppression, and the redundancy constant");
```

Cela donne donc des pistes, la partie principale semblant bien être le fichier `MVirus.nc`. »