RFC 6693 : Probabilistic Routing Protocol for Intermittently Connected Networks

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 1 septembre 2012

Date de publication du RFC : Août 2012

https://www.bortzmeyer.org/6693.html

Les protocoles traditionnels de routage (comme, mettons, OSPF) sont conçus dans l'idée que les machines sont connectées et joignables en permanence, sauf rares pannes. Mais il existe des réseaux où cette hypothèse ne se vérifie pas. Par exemple, dans l'espace, domaine traditionnel du groupe de recherche DTN <http://irtf.org/dtnrg>, il est fréquent que la communication soit l'exception plutôt que la règle. Comment router dans ces conditions ? Ce nouveau RFC propose un mécanisme de routage adapté à des connexions intermittente, Prophet.

On n'est plus dans le monde tranquille de l'Internet, avec ses connexions stables et ses problèmes connus. Ici, on est à l'IRTF, l'équivalent de l'IETF pour les recherches avancées. À l'IRTF, on étudie des problèmes comme les connexions entre vaisseaux perdus dans l'espace, ou groupes d'éleveurs samis se déplaçant en Laponie, ou encore entre objets communiquants se déplaçant sans cesse et ne venant que rarement à portée de radio des autres objets. C'est là que Prophet devient utile. (Voir le RFC 4838 [1] pour avoir une meilleure idée des sujets sur lesquels travaille le groupe DTN.). Les protocoles de routage traditionnels (cf. section 1.3), conçus pour trouver le chemin optimum dans le graphe et pour éviter les boucles, n'ont pas réellement de sens ici (par exemple, l'émetteur et le destinataire peuvent ne jamais être allumés en même temps).

Le principe de base de Prophet est d'utiliser l'**histoire** des rencontres entre deux machines. Si deux engins viennent en contact radio de temps en temps, puis se re-séparent, la mise en œuvre de Prophet se souviendra de ses contacts et pourra se dire « ah, j'ai un "*bundle*" (cf. RFC 5050 et RFC 6255) à passer à Untel et, justement, je l'ai vu trois fois dans la dernière semaine, donc je garde le "*bundle*", je devrais le revoir bientôt. » Au moment du contact, les machines Prophet échangent aussi des informations sur les machines qu'elles ont rencontré donc la décision peut aussi être « ah, j'ai un "*bundle*" à passer à Untel, je

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc4838.txt

ne l'ai jamais croisé mais Machin l'a souvent fait et, justement, j'ai vu trois fois Machin dans la dernière semaine, donc je garde le "*bundle*", je devrais revoir Machin bientôt, et je lui refilerai le "*bundle*" et la responsabilité. »

Une manière courante de résoudre ce problème est d'envoyer le message (dans l'architecture DTN, les messages sont découpés en unités de transmission appelées "*bundles*") à tous ceux qu'on croise, qui le transmettent à leur tour à tous ceux qu'ils croisent et ainsi de suite jusqu'à ce que tout le monde l'ait eu. C'est ce qu'on appelle l'inondation et la mise en œuvre la plus connue de ce principe est dans Usenet (RFC 5537). Mais, dans Usenet, on supposait que tous les nœuds étaient intéressés par le message. Ici, un seul destinataire le veut et, pour ce cas, l'inondation fonctionne mais gaspille pas mal de ressources. Elle marche bien lorsque les ressources ne sont pas un problème mais Prophet se veut utilisable pour des réseaux d'objets communiquants, objets ayant peu de mémoire, de capacité réseau, et de processeur. Prophet adopte donc plutôt une démarche probabiliste : il ne garde que les routes les plus prometteuses, risquant ainsi de ne pas trouver de chemin vers la destination, mais limitant la consommation de ressources. Cet **élagage** des routes est la principale innovation de Prophet.

Pour échanger les routes (pas forcément les "*bundles*"), deux nœuds Prophet, lorsqu'ils se rencontrent, établissent un lien TCP entre eux. La mise en œuvre de Prophet doit donc pouvoir parler TCP et interagir avec le programme de gestion des "*bundles*", pour pouvoir informer les besoins des "*bundles*" qu'on souhaite transmettre.

On notera que, comme toujours dans l'architecture DTN ("*Delay-Tolerant Networking*", cf. RFC 4838), la transmission n'est pas synchrone : le message, découpé en "*bundles*", va être "*bufferisé*" et pourra mettre « un certain temps » à atteindre sa destination. En outre, il est parfaitement possible que la source et la destination du message ne soient jamais allumés et joignables en même temps : on fait donc du réseautage transitif (le message d'Alice est transmis à René, qui le passe à Rose, qui l'envoit finalement au destinataire, Bob). L'annexe A contient un exemple plus détaillé.

À noter que Prophet est utilisé pour des connexions intermittentes, ce qui ne veut pas dire aléatoires. Elles peuvent au contraire être très prévisibles (cas d'engins spatiaux sur leur orbite, ou cas de déplacement d'objets sur un rythme circadien). D'autre part, Prophet n'est pas seulement utile pour le cas où les nœuds se déplacent mais aussi pour celui où ils sont fixes mais souvent en hibernation, par exemple pour économiser du courant (capteur industriel fixé dans un ouvrage d'art, par exemple). Dans les deux cas, la connectivité est intermittente et ce sont les mêmes solutions qu'on utilise. (Voir « "*Epidemic Routing for Partially Connected Ad Hoc Networks*" `<http://issg.cs.duke.edu/epidemic/epidemic.pdf>` »).

Les détails pratiques, maintenant. Le RFC est long et il y a beaucoup de détails à prendre en compte. Ce qui suit est donc un simple résumé. La section 2 présente l'architecture de Prophet. À son cœur, la notion de « probabilité de remise [d'un message] » ("*delivery predictability*"). Notée P_, définie pour tout couple de machines (A, B), comprise entre 0 et 1, c'est la principale métrique de Prophet. Si P_(A, B) ¿ P_(C, B), alors Prophet enverra les "*bundles*" destinés à B via C plutôt que via A. Attention, cette probabilité est asymétrique, P_(A, B) n'a aucune raison d'être égal à P_(B, A). Lorsque deux machines Prophet se rencontrent, elles échangent toutes les probabilités de remise qu'elles connaissent.

Au début, lorsqu'un nœud ne connait rien, la probabilité de remise est indéfinie. À la première rencontre, elle passe typiquement de 0,5 (paramètre `P_encounter_first`), reflétant l'ignorance encore grande. Au fur et à mesure des rencontres, la probabiité augmente (lentement, pour ne pas se heurter au plafond de 1 trop vite, paramètre `delta`). Cette augmentation est amortie par un intervalle minimum entre deux rencontres (paramètre `P_encounter`, pour éviter qu'un lien WiFi très instable ne se traduise par plein de rencontres, qui seraient en fait uniquement des rétablissements du lien). C'est un résumé, la définition complète de ce calcul figure en section 2.1.

Vous allez peut-être vous dire « mais les rencontres du passé ne signifient pas qu'elles vont se reproduire dans le futur ». En effet, si les rencontres sont complètement dues au hasard, le passé n'est pas un bon indicateur. Mais rappelez-vous que, dans les réseaux réels, les rencontres ne sont pas complètement aléatoires.

Ces différents paramètres ne sont pas forcément les mêmes pour tous les nœuds Prophet du réseau mais il est recommandé qu'ils soient proches. La section 3.3 détaille ces paramètres.

Ah, au fait, j'ai dit « quand deux nœuds Prophet se rencontrent ». Mais comment le savent-ils, qu'ils sont en vue l'un de l'autre ? Prophet nécessite que la couche 2 transmette cette information (« *"new neighbor in sight"* »). Comme détaillé en section 2.4, si Prophet peut tourner sur des protocoles de niveau 2 différents, il faut néanmoins que tous soient capables de signaler l'arrivée ou le départ d'un voisin, ainsi que l'adresse de niveau 2 pour le joindre.

La section 3 décrit le protocole utilisé entre les nœuds Prophet. Il permet d'échanger avec le pair :

— La liste des nœuds connus (identifiés par l'EID défini dans le RFC 4838),
— Les probabilités de remise à ces nœuds,
— Les *"bundles"* à envoyer, et la réponse à ceux reçus depuis le pair.

La section 4 présente le format des messages Prophet. Ce format utilise largement des TLV. Chacun a un type (enregistré à l'IANA `<https://www.iana.org/assignments/prophet/prophet.xml#tlv-type>`, cf. section 7) et, par exemple, le TLV `Hello`, qui sert en début de connexion, a le type 1. Les structure de données plus complexes, comme la table de routage, sont encodés en SDNV (RFC 6256).

La section 5 fournit ensuite les machines à états du protocole.

Et la sécurité ? Ce sujet obligatoire est traité en section 6. Ce sera vite fait : Prophet n'a à peu près aucune sécurité. Il est très vulnérable car les exigences du cahier des charges (machines avec peu de ressources, connectées seulement de manière très intermittente) rendent difficile toute conception d'un protocole de sécurité.

Prophet ne fonctionne donc que si tous les nœuds participant sont gentils et œuvrent dans le but commun. Un mécanisme de sécurité simple serait donc que tous les nœuds utilisent une authentification et que seuls les membres d'une même tribu soient autorisés à faire du Prophet ensemble.

Que pourrait-il se passer, autrement ? Un nœud peut prétendre de manière mensongère qu'il a des routes, en mettant toutes les probabilités de remise à 1 (attaque du trou noir : les autres nœuds lui confieront des *"bundles"* qu'il ne pourra pas transmettre). Il peut annoncer un identificateur qui n'est pas le sien, piquant le trafic destiné à un autre. Il peut aussi générer des tas de *"bundles"* et les transmettre à ses malheureux pairs (une attaque par déni de service). Et tout ceci serait encore pire si un réseau Prophet était connecté à l'Internet via une passerelle. Les méchants pourraient alors attaquer l'Internet, cachés derrière les faiblesses de Prophet. Bref, un nœud sadique peut faire beaucoup de dégâts, difficiles à empêcher. D'où l'importance de l'authentification, pour ne jouer qu'avec des copains de confiance.

Les sections 8 et 9 de notre RFC décrivent l'expérience existante avec les mises en œuvre de Prophet. C'est un vieux protocole, bien que le RFC ne sorte que maintenant, et il a déjà été implémenté plusieurs fois. Le première fois, écrite en Java, était pour un environnement original, Lego Mindstorms. Elle est documentée dans la thèse de Luka Birsa `<http://eprints.fri.uni-lj.si/912/>`. Une autre mise en œuvre, en C++, a tourné sur le simulateur OmNet `<http://www.omnetpp.org/>`. Encore une autre, pour Android, est décrite dans l'article « *"Bytewalla 3 : Network architecture and PRoPHET implementation"* `<http://www.bytewalla.org/sites/bytewalla.org/files/Bytewalla3_Network_architecture_and_PRoPHET_v1.0.pdf>` ».

Le tout a déjà été testé dans un environnement réel, dans des montagnes du nord de la Suède, en 2006, et raconté dans « *"Providing connectivity to the Saami nomadic community"* `<http://psg.com/~avri/papers/Saami-Network-Connect-final.pdf>` » (avec une jolie photo de renne dans l'article).