RFC 6126 : The Babel Routing Protocol

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 6 avril 2011

Date de publication du RFC : Avril 2011

https://www.bortzmeyer.org/6126.html

La recherche sur les protocoles de routage ne désarme pas, motivée à la fois par les avancées de la science et par les nouvelles demandes (par exemple pour les réseaux ad hoc). Ainsi Babel est un nouveau protocole de routage, de la famille des protocoles à vecteurs de distance, qui vise notammment à réduire drastiquement les probabilités de boucle. (Depuis, Babel a été normalisé dans le RFC 8966 [1].)

Il y a en effet deux familles de protocole de routage, ceux à vecteurs de distance comme l'ancêtre RIP et ceux à états des liens comme OSPF (RFC 2328). Ce dernier est aujourd'hui bien plus utilisé que RIP, et à juste titre. Mais les problèmes de RIP n'ont pas forcément la même ampleur chez tous les membres de sa famille, et les protocoles à vecteurs de distance n'ont pas dit leur dernier mot.

Babel s'inspire de protocoles de routage plus récents comme DSDV. Il vise à être utilisable, à la fois sur les réseaux classiques, où le routage se fait sur la base du préfixe IP et sur les réseaux ad hoc, où il n'y a typiquement pas de regroupement par préfixe, où le routage se fait sur des adresses IP « à plat » (on peut dire que, dans un réseau ad hoc, chaque nœud est un routeur).

L'un des principaux inconvénients du bon vieux protocole RIP est sa capacité à former des **boucles** lorsque le réseau change de topologie. Ainsi, si un lien entre les routeurs A et B casse, A va envoyer les paquets à un autre routeur C, qui va probablement les renvoyer à A et ainsi de suite (le champ « TTL » pour IPv4 et « "*Hop limit*" » dans IPv6 a précisement pour but d'éviter qu'un paquet ne tourne sans fin). Babel, lui, évitera les boucles la plupart du temps mais, en revanche, il ne trouvera pas immédiatement la route optimale entre deux points. La section 1.1 du RFC spécifie plus rigoureusement les propriétés de Babel.

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc8966.txt

Autre particularité de Babel, les associations entre deux machines pourront se faire même si elles utilisent des paramètres différents (par exemple pour la valeur de l'intervalle de temps entre deux « Hello »; cf. l'annexe B pour une discussion du choix de ces paramètres). Le RFC annonce ainsi que Babel est particulièrement adapté aux environnements « sans-fil » où certaines machines, devant économiser leur batterie, devront choisir des intervalles plus grands.

Je l'ai dit, rien n'est parfait en ce bas monde, et Babel a des limites, décrites en section 1.2. D'abord, Babel envoie périodiquement toutes les informations dont il dispose, ce qui, dans un réseau stable, mène à un trafic total plus important que, par exemple, OSPF (qui n'envoie que les changements). Ensuite, Babel a des mécanismes d'attente lorsqu'un préfixe disparait, qui s'appliquent aux préfixes plus généraux. Ainsi, lorsque deux préfixes deviennent agrégés, l'agrégat n'est pas joignable immédiatement. Notre RFC a le statut « Expérimental » et l'usage découvrira peut-être d'autres faiblesses.

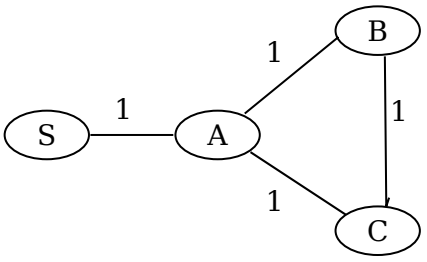
Comment Babel atteint-il ses merveilleux objectifs ? La section 2 détaille les principes de base du protocole, la 3 l'échange de paquets et la 4 l'encodage d'iceux. Commençons par les principes. Babel est fondé sur le bon vieil algorithme de Bellman-Ford, tout comme RIP. Tout lien entre deux points A et B a un **coût** (qui n'est pas forcément un coût monétaire, c'est un nombre qui a la signification qu'on veut, cf. section 3.5.2). Le coût est additif (la somme des coûts d'un chemin complet faisant la **métrique** du chemin, section 2.1 et annexe A), ce qui veut dire que Métrique(A -¿ C) - pour une route passant par B - = Coût(A -¿ B) + Coût(B -¿ C). L'algorithme va essayer de calculer la route ayant la métrique le plus faible.

Un nœud Babel garde trace de ses voisins nœuds en envoyant périodiquement des messages `Hello` et en les prévenant qu'ils ont été entendus par des messages `IHU` ("*I Heard You*"). Le contenu des messages `Hello` et `IHU` permet de déterminer le coût.

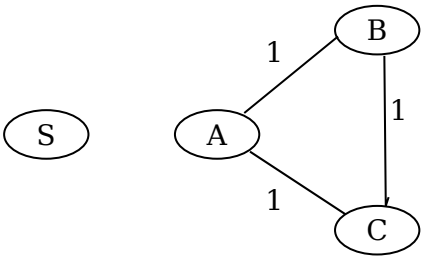
Pour chaque source (d'un préfixe, pas d'un paquet), le nœud garde trace de la métrique vers cette source (lorsqu'un paquet tentera d'atteindre le préfixe annoncé) et du routeur suivant ("*next hop*"). Au début, évidemment la métrique est infinie et le routeur suivant indéterminé. Le nœud envoie à ses voisins les routes qu'il connait. Si celle-ci est meilleure que celle que connait le voisin, ce dernier l'adopte (si la distance était infinie - route inconnue, toute route sera meilleure).

L'algorithme « naïf » ci-dessus est ensuite amélioré de plusieurs façons : envoi immédiat de nouvelles routes (sans attendre l'émission périodique), mémorisation, non seulement de la meilleure route mais aussi de routes alternatives, pour pouvoir réagir plus vite en cas de coupure, etc.

La section 2.3 rappelle un problème archi-connu de l'algorithme de Bellman-Ford : la facilité avec laquelle des boucles se forment. Dans le cas d'un réseau simple comme celui-ci A annonce une route de



métrique 1 vers S, B utilise donc A comme routeur suivant, avec une métrique de 2. Si le lien entre S (S = source de l'annonce) et A casse comme B continue à publier une route de métrique 2 vers S, A se met à envoyer les paquets à B. Mais B les renvoie à A, créant ainsi une boucle. Les annonces ultérieures ne résolvent pas le problème : A annonce une route de métrique 3, passant par B, B l'enregistre et annonce



une route de métrique 4 passant par A, etc. RIP résout le problème en ayant une limite arbitraire à la métrique, limite qui finit par être atteinte et stoppe la boucle (méthode dite du « comptage à l'infini »).

Cette méthode oblige à avoir une limite très basse pour la métrique. Babel a une autre approche : les mises à jour ne sont pas forcément acceptées, Babel teste pour voir si elles créent une boucle (section 2.4). Toute annonce est donc examinée au regard d'une **condition**, dite « de faisabilité ». Plusieurs conditions sont possibles. Par exemple, BGP utilise la condition « Mon propre numéro d'AS n'apparaît pas dans l'annonce. ». (Cela n'empêche pas les micro-boucles, boucles de courte durée en cas de coupure, cf. RFC 5715.) Une autre condition, utilisée par DSDV et AODV, repose sur l'observation qu'une boucle ne se forme que lorsqu'une annonce a une métrique supérieure à la métrique de la route qui a été retirée. En n'acceptant que les annonces qui diminuent la métrique, on peut donc éviter les boucles. Babel utilise une règle un peu plus complexe, empruntée à EIGRP, qui tient compte de l'histoire des annonces faites par le routeur.

Comme il n'y a pas de miracles en routage, cette idée de ne pas accepter n'importe quelle annonce de route a une contrepartie : la famine. Celle-ci peut se produire lorsqu'il existe une route mais qu'aucun routeur ne l'accepte (section 2.5). EIGRP résout le problème en « rédémarrant » tout le réseau (resynchronisation globale des routeurs). Babel, lui, emprunte à DSDV une solution moins radicale, en numérotant les annonces, de manière strictement croissante, lorsqu'un routeur détecte un changement dans ses liens. Une route pourra alors acceptée si elle est plus récente (si elle a un numéro de séquence plus élevé).

À noter que tout se complique s'il existe plusieurs routeurs qui annoncent originellement la même route (section 2.7 ; un exemple typique est la route par défaut, annoncée par tous les routeurs ayant une connexion extérieure). Babel gère ce problème en associant à chaque préfixe l'identité du routeur qui s'est annoncé comme origine et considère par la suite ces annonces comme distinctes, même si le préfixe est le même. Conséquence : Babel ne peut plus garantir qu'il n'y aura pas de boucle (Babel essaie de construire un graphe acyclique mais l'union de plusieurs graphes acycliques n'est pas forcément acyclique). Par contre, il pourra détecter ces boucles a posteriori et les éliminer plus rapidement qu'avec du comptage vers l'infini.

Voilà pour les principes. Et le protocole ? La section 3 le décrit. Chaque routeur a une identité sur huit octets (le plus simple est de prendre l'adresse MAC d'une des interfaces). Les messages sont envoyés dans des paquets UDP et encodés en TLV. Le paquet peut être adressé à une destination "*unicast*" ou bien "*multicast*".

Un routeur Babel doit se souvenir d'un certain nombre de choses (section 3.2) :
— Le numéro de séquence, qui croît strictement,
— La liste des interfaces réseau où parler le protocole,
— La liste des voisins qu'on a entendus,
— La liste des sources (routeurs qui ont été à l'origine de l'annonce d'un préfixe). Elle sert pour calculer les critères d'acceptation (ou de rejet) d'une route. Babel consomme donc plus de mémoire que RIP, qui ne connait que son environnement immédiat, alors qu'un routeur Babel connaît tous les routeurs du réseau.

— Et bien sûr la table des routes, celle qui, au bout du compte, sera utilisée pour la transmission des paquets.

Les messages Babel ne bénéficient pas d'une garantie de délivrance (c'est de l'UDP, après tout), mais un routeur Babel peut demander à ses voisins d'accuser réception (section 3.3). La décision de le demander ou pas découle de la politique locale de chaque routeur. Si un routeur ne demande pas d'accusé de réception, l'envoi périodique des routes permettra de s'assurer que, au bout d'un certain temps, tous les routeurs auront toute l'information. Les accusés de réception peuvent toutefois être utiles en cas de mises à jour urgentes dont on veut être sûr qu'elles ont été reçues. (L'implémentation actuelle de Babel ne les utilise pas.)

Comment un nœud Babel trouve t-il ses voisins ? La section 3.4 décrit ce mécanisme. Les voisins sont détectés par les messages Hello qu'ils émettent. Les messages IHU ("*I Heard You*") envoyés en sens inverse permettent notamment de s'assurer que le lien est bien bidirectionnel.

Les détails de la maintenance de la table de routage figurent en section 3.5. Chaque mise à jour envoyée par un nœud Babel est un quintuplet {préfixe IP, longueur du préfixe, ID du routeur, numéro de séquence, métrique}. Chacune de ces mises à jour est évaluée en regard des conditions de faisabilité : une distance de faisabilité est un doublet {numéro de séquence, métrique} et ces distances sont ordonnées en comparant d'abord le numéro de séquence (numéro plus élevée =¿ distance de faisabilité meilleure) et ensuite la métrique (où le critère est inverse). Une mise à jour n'est acceptée que si sa distance de faisabilité est meilleure.

Si la table des routes contient plusieurs routes vers un préfixe donné, laquelle choisir et donc réannoncer aux voisins (section 3.6) ? La politique de sélection n'est pas partie intégrante de Babel. Plusieurs mises en œuvre de ce protocole pourraient faire des choix différents. Les seules contraintes à cette politique sont qu'il ne faut jamais réannoncer les routes avec une métrique infinie (ce sont les retraits, lorsqu'une route n'est plus accessible), ou les routes infaisables (selon le critère de faisabilité cité plus haut). Si les différents routeurs ont des politiques différentes, cela peut mener à des oscillations (routes changeant en permanence) mais il n'existe pas à l'heure actuelle de critères scientifiques pour choisir une bonne politique. On pourrait imaginer que le routeur ne garde que la route avec la métrique le plus faible, ou bien qu'il privilégie la stabilité en gardant la première route sélectionnée, ou encore qu'il prenne en compte des critères comme la stabilité du routeur voisin dans le temps. En attendant les recherches sur ce point, la stratégie conseillée est de privilégier la route de plus faible métrique, en ajoutant un petit délai pour éviter de changer trop souvent.

Une fois le routeur décidé, il doit envoyer les mises à jour à ses voisins (section 3.7). Ces mises à jour sont transportées dans des paquets "*multicast*" (mais peuvent l'être en "*unicast*"). Les changements récents sont transmis immédiatement, mais un nœud Babel transmet de toute façon la totalité de ses routes à intervalles réguliers. Petite optimisation : les mises à jour ne sont pas transmises sur l'interface réseau d'où la route venait, **mais uniquement** si on est sûr que ladite interface mène à un réseau symétrique (un Ethernet filaire est symétrique mais un lien WiFi ad hoc ne l'est pas forcément).

Un routeur Babel peut toujours demander explicitement des annonces de routes à un voisin (section 3.8). Il peut aussi demander une incrémentation du numéro de séquence, au cas où il n'existe plus aucune route pour un préfixe donné (problème de la famine, section 3.8.2.1).

La section 4 spécifie l'encodage des messages Babel sur le réseau. C'est un paquet UDP, envoyé à une adresse "*multicast*" (`FF02:0:0:0:0:0:1:6` ou `224.0.0.111`) ou bien "*unicast*", avec un TTL de 1 (puisque les messages Babel n'ont jamais besoin d'être routés), et un port source et destination de 6696. En IPv6, les adresses IP de source et de destination "*unicast*" sont "*link-local*" et en IPv4 des adresses du réseau local.

Les données envoyées dans le message sont typées et la section 4.1 liste les types possibles, par exemple "*interval*", un entier de 16 bits qui sert à représenter des durées en centisecondes (rappelez-vous que, dans Babel, un routeur informe ses voisins de ses paramètres temporels, par exemple de la fréquence à laquelle il envoie des `Hello`). Plus complexe est le type "*address*", puisque Babel permet d'encoder les adresses par différents moyens (par exemple, pour une adresse IPv6 "*link-local*", le préfixe `fe80::/64` peut être omis).

Ensuite, ces données sont mises dans des TLV, eux-même placés derrière l'en-tête Babel, qui indique un nombre magique (42...) pour identifier un paquet Babel, un numéro de version (aujourd'hui 2) et la longueur du message. (La fonction `babel_print_v1` dans le "*patch*" de tcpdump est un bon moyen de découvrir les différents types et leur rôle.) Chaque TLV, comme son nom l'indique, comprend un type (entier sur huit bits), une longueur et une valeur, le **corps**, qui peut comprendre plusieurs champs (dépendant du type). Parmi les types existants :

— 0 et 1, qui doivent être ignorés (ils servent si on a besoin d'aligner les TLV),
— 2, qui indique une demande d'accusé de réception, comme le "*Echo Request*" d'ICMP (celui qui est utilisé par la commande ping). Le récepteur doit répondre par un message contenant un TLV de type 3.
— 4, qui désigne un message `Hello`. Le corps contient notamment le numéro de séquence actuel du routeur. Le type 5 désigne une réponse au `Hello`, le IHU, et ajoute des informations comme le coût de la liaison entre les deux routeurs ou comme la liste des voisins entendus (les amateurs d'OSPF noteront que, dans ce protocole, la liste des voisins détectés apparaît dans le `Hello`, ce qui augmente sa taille).
— 6 sert pour transmettre l'ID du routeur.
— 7 et 8 servent pour les routes elles-mêmes. 7 désigne le routeur suivant qui sera utilisé ("*next hop*") pour les routes portées dans les TLV de type 8. Chaque TLV `Update` (type 8) contient notamment un préfixe (avec sa longueur), un numéro de séquence, et une métrique.
— 9 est une demande explicite de route (lorsqu'un routeur n'a plus de route vers un préfixe donné ou simplement lorsqu'il est pressé et ne veut pas attendre le prochain message). 10 est la demande d'un nouveau numéro de séquence.

Quelle est la sécurité de Babel ? La section 6 dit franchement qu'elle est à peu près inexistante. Un méchant peut annoncer les préfixes qu'il veut, avec une faible métrique pour être sûr d'être sélectionné, afin d'attirer tout le trafic. Des extensions futures à Babel (par exemple telles que spécifiées dans le projet `draft-ietf-ospf-auth-trailer-ospfv3`) permettront peut-être de signer les messages mais ce n'est pas encore fait. (Notons que, en matière de routage, la signature ne résout pas tout : c'est une chose d'authentifier un voisin, une autre de vérifier qu'il est autorisé à annoncer ce préfixe.)

En IPv6, une protection modérée est fournie par le fait que les adresses source et destination sont locales au lien. Comme les routeurs IPv6 ne sont pas censés faire suivre les paquets ayant ces adresses, cela garantit que le paquet vient bien du réseau local.

Vous pourrez trouver plus d'informations sur Babel en lisant le RFC, ou bien sur la page Web officielle `<http://www.pps.jussieu.fr/~jch/software/babel/>`.

Qu'en est-il des mises enœuvre de ce protocole ? Il existe une implémentation d'exemple `<http://www.pps.jussieu.fr/~jch/software/babel/>`, assez éprouvée pour être disponible en paquetage dans plusieurs systèmes, comme babeld `<http://packages.debian.org/babeld>` dans Debian ou dans OpenWrt, plateforme très souvent utilisée pour des routeurs libres (cf. `<https://dev.openwrt.org/browser/packages/net/babel>`). Si vous voulez écrire votre implémentation, l'annexe C contient plusieurs conseils utiles, accompagnés de calculs, par exemple sur la consommation mémoire et réseau. Le RFC proclame que Babel est un protocole relativement simple et, par exemple, l'implémentation de référence contient environ 7300 lignes de C.

Néanmoins, cela peut être trop, une fois compilé, pour des objets (le RFC cite les fours à micro-ondes...) et l'annexe C décrit donc des sous-ensembles raisonnables de Babel. Par exemple, une mise en œuvre passive pourrait apprendre des routes, sans rien annoncer. Plus utile, une mise en œuvre « parasite » n'annonce que la route vers elle-même et ne retransmet pas les routes apprises. Ne routant les paquets, elle ne risquerait pas de créer des boucles et pourrait donc omettre certaines données, comme la liste des sources. (Le RFC liste par contre ce que la mise en œuvre parasite **doit** faire.)

Toujours côté programmes, il existe un "*patch*" à tcpdump pour afficher les paquets Babel et un en cours <https://bugs.wireshark.org/bugzilla/show_bug.cgi?id=5812> pour Wireshark. On commence à voir des traces Babel <http://www.pcapr.net/browse?q=babel> sur pcapr.net <https://www.bortzmeyer.org/pcapr.html>.

Si vous voulez approfondir la question des protocoles de routage, une excellente comparaison a été faite par David Murray, Michael Dixon et Terry Koziniec dans « "*An Experimental Comparison of Routing Protocols in Multi Hop Ad Hoc Networks*" <http://researchrepository.murdoch.edu.au/3982/1/Comparison_of_Routing_Protocols.pdf> » où ils comparent Babel (qui l'emporte largement), OLSR (RFC 7181) et Batman (ce dernier est dans le noyau Linux officiel). Notez aussi que l'IETF a un protocole standard pour ce problème, RPL, décrit dans le RFC 6550.

Merci beaucoup à Juliusz Chroboczek pour sa relecture et ses nombreux avis.