RFC 8966 : The Babel Routing Protocol

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 12 janvier 2021

Date de publication du RFC : Janvier 2021

https://www.bortzmeyer.org/8966.html

Le travail sur les protocoles de routage ne désarme pas, motivé 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 protocole de routage, de la famille des protocoles à vecteur de distance, qui vise notammment à réduire drastiquement les probabilités de boucle. Il avait été décrit originellement dans le RFC 6126 [1] ; ce nouveau RFC ne change pas fondamentalement le protocole Babel (quoique certains changements ne seront pas compris par les vieilles versions) mais il a désormais le statut de norme, et des solutions de sécurité plus riches. Ce manque de sécurité était la principale critique adressée à Babel.

Il y a deux familles de protocole de routage, ceux à vecteur 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

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc6126.txt

immédiatement la route optimale entre deux points. La section 1.1 du RFC spécifie plus rigoureusement les propriétés de Babel.

Babel peut fonctionner avec différentes métriques pour indiquer les coûts de telle ou telle route, le protocole lui-même étant indépendant de la métrique utilisée (cf. annexe A du RFC pour des conseils sur les choix). D'ailleurs, je vous recommande la lecture de l'"*Internet-Draft*" `draft-chroboczek-babel-doesnt-care` pour mieux comprendre la philosophie de Babel.

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, les compromis que ce choix implique entre intensité du trafic et détection rapide des changements, et les valeurs recommandées pour 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, ou bien aux environnements non gérés, où chaque machine est configurée indépendamment.

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.

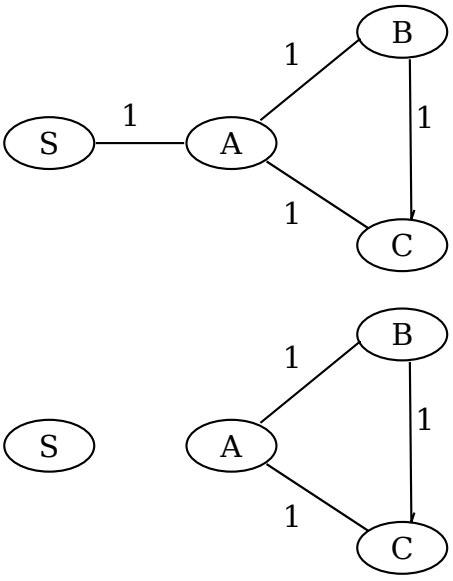
Comment Babel atteint-il ses merveilleux objectifs ? La section 2 détaille les principes de base du protocole, la 3 l'échange de paquets et la 4 l'encodage d'iceux. Commençons par les principes. Babel est fondé sur le bon vieil algorithme de Bellman-Ford, tout comme RIP. Tout lien entre deux points A 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 (RFC 3561), repose sur l'observation qu'une boucle ne se forme que lorsqu'une annonce a une métrique moins bonne que la métrique de la route qui a été retirée. En n'acceptant que les annonces qui améliorent 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 être acceptée si elle est plus récente (si elle a un numéro de séquence plus élevé), et un routeur Babel peut demander explicitement aux autres routeurs d'incrémenter ce nombre, pour accélérer la convergence. Ce numéro n'est par contre pas utilisé pour sélectionner la meilleure route (seule la métrique compte pour cela), uniquement pour voir si une annonce est récente.

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.

Notez aussi que chaque routeur Babel est libre de rejeter les routes qui lui semblent déraisonnables, comme `127.0.0.1/32`, sans affecter le fonctionnement du protocole (le détail de cette question du filtrage des routes est dans l'annexe C.)

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*". Les TLV peuvent contenir des sous-TLV dans leur partie Valeur.

Un routeur Babel doit se souvenir d'un certain nombre de choses (section 3.2), notamment :
— 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 préfixes annoncés sont sans rapport avec la version du protocole IP utilisée pour transporter l'annonce. Un préfixe IPv4 peut donc être envoyé en IPv6. Le RFC recommande de faire tourner Babel sur IPv6, même si le réseau est en partie en IPv4.

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.

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. Notez que la méthode de calcul des métriques n'est pas imposée par Babel : tant que cette méthode obéit à certains critères, notamment de monotonie, elle peut être utilisée.

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::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 <`https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xml`>. En IPv6, les adresses IP de source et de destination "*unicast*" sont locales au lien 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 locale au lien, le préfixe `fe80::/64` peut être omis). Quant à l'ID du routeur, cet identifiant est stocké sur huit octets.

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_v2` dans le code de tcpdump est un bon moyen de découvrir les différents types et leur rôle.) Le message est suivi d'une remorque qui n'est pas comptée pour le calcul de la longueur du message, et qui sert notamment pour l'authentification (cf. RFC 8967). La remorque, une nouveauté qui n'existait pas explicitement dans le RFC 6126, est elle-même composée de TLV. 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.
— 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.

Les types de TLV sont stockés dans un registre IANA <https://www.iana.org/assignments/babel/babel.xml#tlv-types>. On peut en ajouter à condition de fournir une spécification écrite (« Spécification nécessaire », cf. RFC 8126). Il y a aussi un registre des sous-TLV <https://www.iana.org/assignments/babel/babel.xml#sub-tlv-types>.

Quelle est la sécurité de Babel ? La section 6 dit franchement qu'elle est, par défaut, à peu près inexistante. Un méchant peut annoncer les préfixes qu'il veut, avec une bonne métrique pour être sûr d'être sélectionné, afin d'attirer tout le trafic.

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. Une raison de plus d'utiliser IPv6.

Ce manque de sécurité dans le Babel original du RFC 6126 avait suscité beaucoup de discussions à l'IETF lors de la mise de Babel sur le chemin des normes (voir par exemple cet examen de la direction de la sécurité <https://datatracker.ietf.org/doc/review-ietf-babel-rfc6126bis-10-secdir-lc-ka>). Normalement, l'IETF exige de tout protocole qu'il soit raisonnablement sécurisé (la règle figure dans le RFC 3365). Certaines personnes s'étaient donc vigoureusement opposées à la normalisation officielle de Babel tant qu'il n'avait pas de solution de sécurité disponible. D'autres faisaient remarquer que Babel était quand même déployable pour des réseaux fermés, « entre copains », même sans sécurité, mais les critiques pointaient le fait que tôt ou tard, tout réseau fermé risque de se retrouver ouvert. D'un autre côté, sécuriser des réseaux « ad hoc », par exemple un lot de machines mobiles qui se retrouvent ensemble pour un événement temporaire, est un problème non encore résolu.

Un grand changement de notre RFC est de permettre la signature des messages. Deux mécanismes existent, décrits dans les RFC 8967 (signature HMAC, pour authentifier, la solution la plus légère) et RFC 8968 (DTLS, qui fournit en plus la confidentialité). (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.)

J'ai parlé plus haut de la détermination des coûts des liens. L'annexe A du RFC contient des détails intéressants sur cette détermination. Ainsi, contrairement aux liens fixes, les liens radio ont en général une qualité variable, surtout en plein air. Déterminer cette qualité est indispensable pour router sur des liens radio, mais pas facile. L'algorithme ETX (décrit dans l'article de De Couto, D., Aguayo, D., Bicket, J., et R. Morris, « "*A high-throughput path metric for multi-hop wireless networks*" <https://dl.acm.org/doi/abs/10.1145/938985.939000> ») est recommandé pour cela.

L'annexe D est consacrée aux mécanismes d'extension du protocole, et reprend largement le RFC 7557, qu'elle remplace. Babel prévoyait des mécanismes d'extension en réservant certaines valeurs et en précisant le comportement d'un routeur Babel lors de la réception de valeurs inconnues. Ainsi :

— Un paquet Babel avec un numéro de version différent de 2 doit être ignoré, ce qui permet de déployer une nouvelle future version de Babel sans que ses paquets ne cassent les implémentations existantes,
— Un TLV de type inconnu doit être ignoré (section 4.3), ce qui permet d'introduire de nouveaux types de TLV en étant sûr qu'ils ne vont pas perturber les anciens routeurs,
— Les données contenues dans un TLV au-delà de sa longueur indiquée, ou bien les données présentes après le dernier TLV, devaient, disait le RFC 7557 qui reprenait le RFC 6126, également être silencieusement ignorées (au lieu de déclencher une erreur). Ainsi, une autre voie d'extension était possible, pour glisser des données supplémentaires. Cette voie est désormais utilisée par les solutions de signature comme celle du RFC 8966.

Quelles sont donc les possibilités d'extensions propres ? Cela commence par une nouvelle version du protocole (l'actuelle version est la 2), qui utiliserait des numéros 3, puis 4... Cela ne doit être utilisé que si la nouvelle version est incompatible avec les précédentes et ne peut pas interopérer sur le même réseau.

Moins radicale, une extension de la version 2 peut introduire de nouveaux TLV (qui seront ignorés par les mises en œuvre anciennes de la version 2). Ces nouveaux TLV doivent suivre le format de la section 4.3. De la même façon, on peut introduire de nouveaux sous-TLV (des TLV contenus dans d'autres TLV, décrits en section 4.4).

Si on veut ajouter des données dans un TLV existant, en s'assurant qu'il restera correctement analysé par les anciennes mises en œuvre, il faut jouer sur la différence entre la taille explicite ("*explicit size*") et la taille effective ("*natural size*") du TLV. La taille explicite est celle qui est indiqué dans le champ `Length` spécifié dans la section 4.3. La taille effective est celle déduite d'une analyse des données (plus ou moins compliquée selon le type de TLV). Comme les implémentations de Babel doivent ignorer les données situées après la taille explicite, on peut s'en servir pour ajouter des données. Elles doivent être encodées sous forme de sous-TLV, chacun ayant type, longueur et valeur (leur format exact est décrit en section 4.4).

Enfin, après le dernier TLV (Babel est transporté sur UDP, qui indique une longueur explicite), on peut encore ajouter des données, une « remorque ». C'est ce que fait le RFC 8966.

Bon, mais alors quel mécanisme d'extension choisir ? La section 4 fournit des pistes aux développeurs. Le choix de faire une nouvelle version est un choix drastique. Il ne devrait être fait que si la nouvelle version est réellement incompatible avec la précédente.

Un nouveau TLV, ou bien un nouveau sous-TLV d'un TLV existant est la solution à la plupart des problèmes d'extension. Par exemple, si on veut mettre de l'information supplémentaire aux mises à jour de routes (TLV `Update`), on peut créer un nouveau TLV « `Update` enrichi » ou bien un sous-TLV de `Update` qui contiendra l'information supplémentaire. Attention, les conséquences de l'un ou l'autre choix ne seront pas les mêmes. Un TLV « `Update` enrichi » serait totalement ignoré par un Babel ancien, alors qu'un TLV `Update` avec un sous-TLV d'« enrichissement » verrait la mise à jour des routes acceptée, seule l'information supplémentaire serait perdue.

Il existe désormais, pour permettre le développement d'extensions, un registre IANA des types de TLV `<https://www.iana.org/assignments/babel/babel.xhtml#tlv-types>` et un des sous-TLV `<https://www.iana.org/assignments/babel/babel.xhtml#sub-tlv-types>` (section 5 du RFC) et plusieurs extensions s'en servent déjà.

Enfin, l'annexe F du RFC résume les changements depuis le premier RFC, le RFC 6126, qui documentait la version 2 de Babel. On reste en version 2 car le protocole de notre RFC reste essentiellement compatible avec le précédent. Il y a toutefois trois fonctions de ce protocole qui pourraient créer des problèmes sur un réseau où certaines machines sont restées au RFC 6126 :

— Les messages de type `Hello` en "*unicast*" sont une nouveauté. L'ancien RFC ne les mentionnait pas. Cela peut entrainer une mauvaise interprétation des numéros de séquence (qui sont distincts en "*unicast*" et "*multicast*").
— Les messages `Hello` peuvent désormais avoir un intervalle entre deux messages qui est nul, ce qui n'existait pas avant.
— Les sous-TLV obligatoires (section 4.4) n'existaient pas avant et leur utilisation peut donc être mal interprétée par les vieilles mises en œuvre de Babel (le TLV englobant va être accepté alors qu'il devrait être rejeté).

Bref, si on veut déployer le Babel de ce RFC dans un réseau où il reste de vieilles mises en œuvre, il faut prendre garde à ne pas utiliser ces trois fonctions. Si on préfère mettre à jour les vieux programmes, sans toutefois y intégrer tout ce que contient notre RFC, il faut au minimum ignorer (ou bien gérer correctement) les `Hello` en "*unicast*", ou bien avec un intervalle nul, et il faut ignorer un TLV qui contient un sous-TLV obligatoire mais inconnu.

Il y a d'autres changements depuis le RFC 6126 mais qui ne sont pas de nature à affecter l'interopérabilité ; voyez le RFC pour les détails.

Vous pourrez trouver plus d'informations sur Babel en lisant le RFC, ou bien sur la page Web officielle <https://www.irif.fr/~jch//software/babel/>. 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*" <https://researchrepository.murdoch.edu.au/id/eprint/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. Si vous aimez les vidéos, l'auteur de Babel explique le protocole en anglais <https://www.youtube.com/watch?v=SMJoQolFnOg>.

Qu'en est-il des mises en œuvre de ce protocole ? Il existe une implémentation d'exemple <https://www.irif.fr/~jch//software/babel/>, babeld <https://github.com/jech/babeld>, assez éprouvée pour être disponible en paquetage dans plusieurs systèmes, comme babeld <https://packages.debian.org/babeld> dans Debian ou dans OpenWrt, plateforme très souvent utilisée pour des routeurs libres (cf. <https://openwrt.org/docs/guide-user/services/babeld>). babeld ne met pas en œuvre les solutions de sécurité des RFC 8966 ou RFC 8967. Une autre implémentation se trouve dans Bird. Si vous voulez écrire votre implémentation, l'annexe E 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 12 500 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 E 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, tcpdump et Wireshark savent afficher les paquets Babel.