Projet de programmation réseau

Juliusz Chroboczek

19 novembre 2016

1 Introduction

Le but de ce projet est d'implémenter un protocole d'inondation similaire à celui qui est utilisé par les protocoles de routage à état de lien (OSPF ou IS-IS). À la différence de ces derniers, il fonctionne à travers l'Internet, et permet de synchroniser une petite quantité de données à travers un nombre modéré de pairs. Le protocole est défini précisément par ce document. Les implémentations non interopérables ne seront pas acceptées.

2 Description générale du protocole

Chaque nœud est identifié par un *Id* globalement unique.

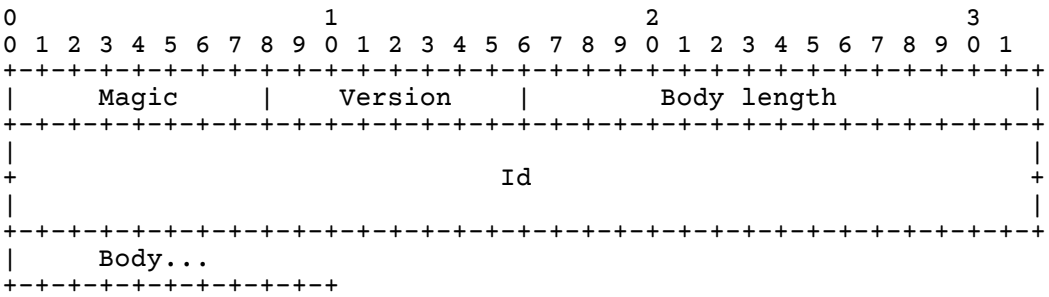
Chaque nœud maintient une liste de voisins (paragraphe 4.2) — ce sont des nœuds avec lesquels il a déterminé que la connectivité symétriquele est possible. Initialement, un nœud contacte un petit nombre de *nœuds de bootstrap* bien connus ; par la suite, il repeuple sa liste de voisins en demandant à ses voisins les adresses de leurs voisins et ainsi de suite.

Chaque nœud publie des données sous un format bien défini (paragraphe 3.3). Chaque version des données que le nœud publie est accompagnée d'un *numéro de séquence* qui permet aux autres nœuds de déterminer la version la plus récente.

Finalement, chaque nœud participe à un protocole d'inondation fiable qui permet de synchroniser les données à travers le réseau.

3 Format de paquets

Tous les messages envoyés par ce protocole sont encapsulés dans un paquet UDP. Chaque pair doit être capable de recevoir des paquets UDP (fragmentés) d'une taille inférieure ou égale à 4096 octets (entête inclus). Chaque paquet est constitué d'un entête suivi d'une suite de messages. L'entête a le format suivant :



Les champs sont définis comme suit :

Magic : cet octet vaut 57. Tout paquet qui ne commence pas par un octet valant 57 sera ignoré par le récepteur.

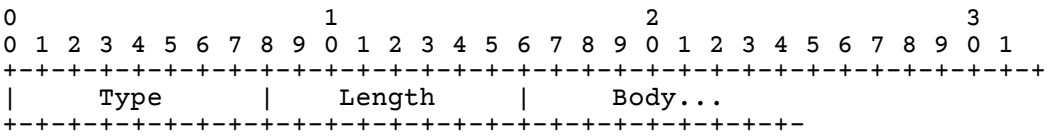
Version : cet octet vaut 0. Tout paquet qui ne contient pas un champ version valant 0 sera ignoré par le récepteur.

Body length : ce champ indique la longueur des données qui suivent (sans compter les champs *Magic*, *Version*, *Body length* et *Id*)). Si la charge du paquet UDP est plus grande que *Body Length* + 24, les données supplémentaires sont ignorées.

Id : le *Id* de l'émetteur.

3.1 Format des messages

Le corps d'un paquet (le champ *Packet Body* du diagramme précédent) est composé d'une suite de TLV, c'est à dire de message constitués d'un *Type*, d'une *Longueur*, et d'une *Valeur*. À l'exception du TLV *Pad1*, chaque TLV a la forme suivante :



Les champs sont définis comme suit :

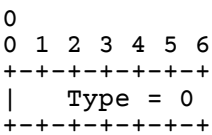
Type : le type du TLV, un entier.

Length : la longueur du corps, sans compter les champs *Type* et *Length* ;

Body : une suite d'octets de longueur *Length*, dont l'interprétation dépend du TLV.

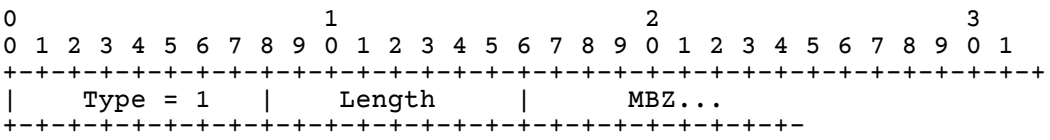
3.2 Détails des messages

Pad1



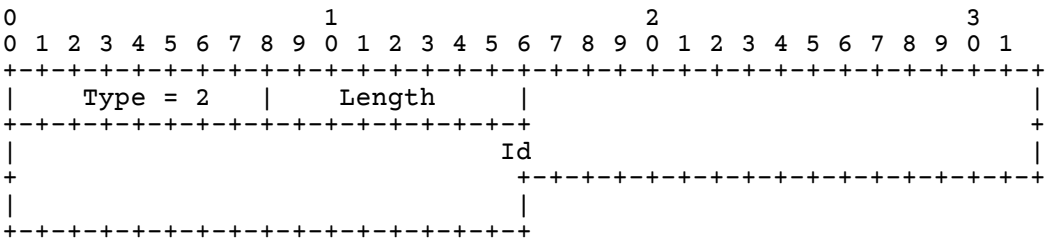
Ce TLV est ignoré à la réception.

PadN



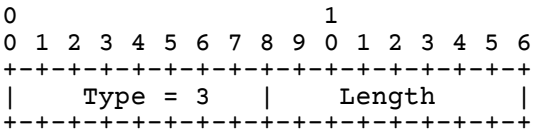
Ce TLV est ignoré à la réception. Le champ MBZ est une suite de zéros.

IHU (I Heard You)



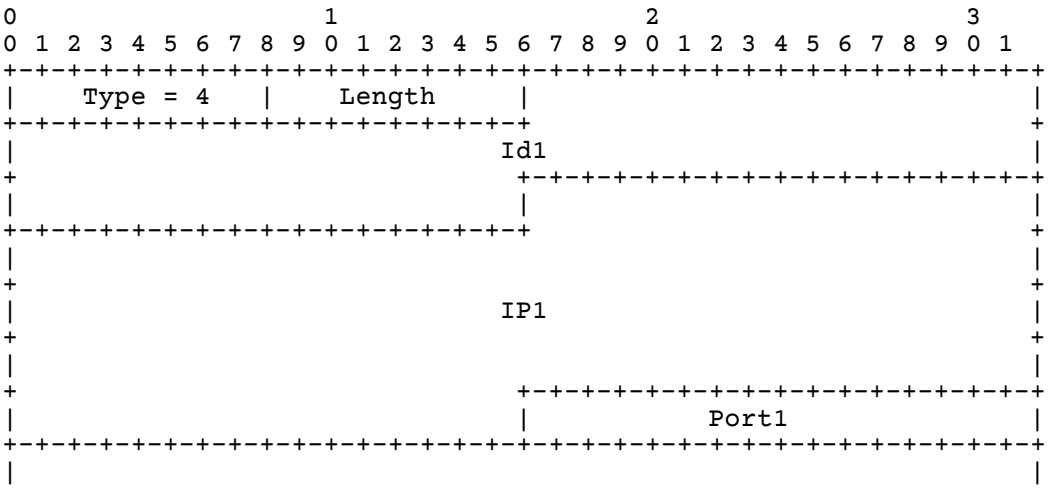
Ce TLV est envoyé à chaque voisin toutes les 90 secondes environ. Le champ *Id* contient l'*Id* du voisin. Si le corps contient des données après le champ *Id* (i.e. *Length* > 8), elles sont ignorées.

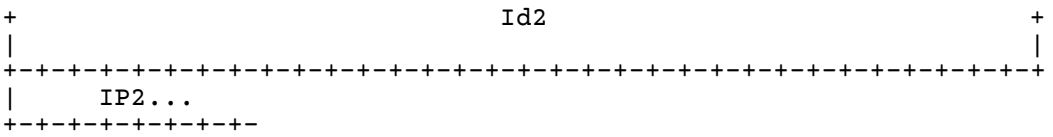
Neighbour Request



Ce TLV demande une liste de nouveaux pairs (voir ci-dessous). Si le corps contient des données (i.e. *Length* > 0), elles sont ignorées.

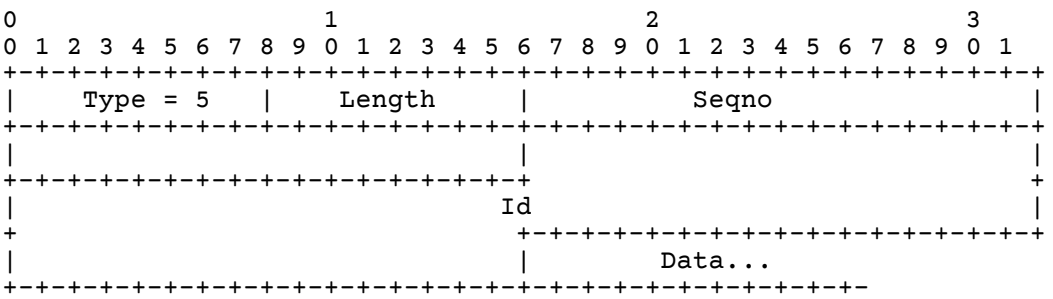
Neighbours





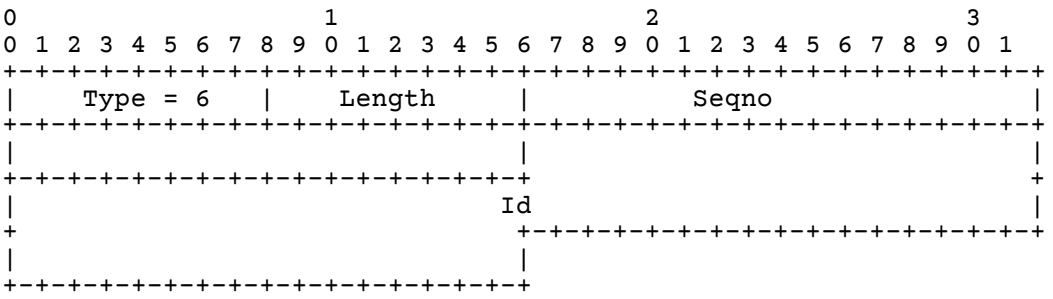
Ce TLV contient une partie de la liste de voisins symétriques de l'émetteur, représentée par des triplets (Id, IP, Port). Les adresses IPv6 sont représentées telles quelles, les adresses IPv4 sont représentées sous forme *IPv4-Mapped* (dans le préfixe `::FFFF:0:0/96`). Il est normalement envoyé en réponse à un TLV *Neighbour-Request*, mais peut aussi être envoyé spontanément. Le champ *Length* doit être un multiple de 8.

Data



Ce TLV sert à inonder les données. Le champ *Seqno* est un entier de 32 bits, incrémenté par l'originateur à chaque fois que de nouvelles données sont publiées. Le champ *Id* est l'Id du nœud qui a publié ces données. Le champ *Data* est décrit au paragraphe 3.3.

IHave



Ce TLV sert à indiquer à un pair que l'émetteur possède la donnée indiquée par *Seqno* et *Id*.

Autres TLV Si le paquet contient des TLV autres que ceux définis ci-dessus, ils sont ignorés (mais les autres TLV du paquet sont interprétés normalement).

3.3 Données

Lors de l'inondation, le champ *Data* n'est pas interprété, mais traité comme une chaîne opaque : il n'est pas correct de l'analyser avant de l'inonder, il faut l'inonder tel quel même si on n'a pas pu l'analyser.

Cependant, il est normalement structuré comme une suite de TLV. Pour le moment, je définis les TLV de données suivants :

- *Type* = 32 : une chaîne de caractères dans le codage UTF-8 ;
- *Type* = 33 : une image au format PNG ;
- *Type* = 34 : une image au format JPEG.

Les autres TLV sont autorisés, et votre implémentation devra inonder correctement les données qu'elle ne sait pas analyser. Si vous voulez publier d'autres données, prenez un numéro de TLV supérieur à 34, mais coordonnez-vous avec moi pour éviter les collisions.

4 Description détaillée du protocole de base

4.1 Structures de données

Dans ce paragraphe, je décris les structures de données *conceptuelles* de chaque pair. *Conceptuelles* signifie qu'une implémentation différente est possible — par exemple, au lieu d'avoir trois listes de pairs, il est possible d'utiliser une seule liste avec des *flags* indiquant la nature du pair, etc.

Id Chaque pair se choisit un *Id*, une chaîne de 64 bits supposée globalement unique. On peut par exemple la tirer au hasard lors du démarrage, et optionnellement le stocker dans un fichier pour conserver le même *Id* lors de redémarrage.

Seqno Chaque pair maintient un *Seqno*, un entier croissant de 32 bits qui indique la version des données qu'il publie. Cet entier doit être incrémenté au moins une fois toutes les 30 minutes. On peut soit utiliser un entier qu'on incrémente à chaque fois qu'on change les données publiées (ce qui peut poser des problèmes au redémarrage), ou alors un temps depuis une origine arbitraire.

Données Chaque pair maintient une table de données, des triplets de la forme *Id*, *Seqno*, *Data*, ainsi que la date à laquelle cette donnée a été vue pour la première fois. Initialement, les données consistent seulement des données publiées par le pair local ; la table est peuplée lors de l'inondation.

Listes de voisins Chaque pair maintient trois listes (disjointes) de pairs :

- la liste des voisins potentiels ;
- la liste des voisins unidirectionnels ;
- la liste des voisins symétriques.

Un voisin potentiel consiste d'une adresse IP qu'on peut éventuellement contacter pour déterminer s'il s'agit d'un pair. La liste des voisins unidirectionnels contient les voisins dont on a reçu un paquet dans les dernières 100 secondes ; elle contient donc pour chaque pair la date du dernier paquet reçu. La liste des voisins symétriques contient les voisins dont on a reçu un paquet dans les dernières 150 secondes, et un *IHU* avec l'Id du pair local dans les dernières 300 secondes ; elle contient donc pour chaque pair la date du dernier paquet reçu et du dernier *IHU* reçu.

4.2 Maintenance de la liste de voisins

Initialement, la liste de voisins potentiels est peuplée d'un petit nombre de *nœuds de bootstrap*. Les autres listes sont vides.

Toutes les 30 secondes environ, un pair envoie un paquet vide (avec seulement un entête) à chaque pair de sa liste de voisins unidirectionnels et symétriques. De plus, s'il a moins de 5 voisins symétriques, il envoie un paquet vide à un de ses pairs potentiels (tiré au hasard).

Toutes les 90 secondes environ, un pair envoie un paquet contenant un TLV *IHU* à chacun de ses voisins unidirectionnels et symétriques.

Lorsqu'un pair *A* reçoit un paquet d'un pair *B*, et *B* n'est pas un voisin unidirectionnel ou symétrique, il supprime le cas échéant *B* de la liste de voisisn potentiels, et ajoute *B* à sa liste de voisins unidirectionnels. Dans tous les cas, il met à jour la date du dernier paquet reçu de *B*.

Lorsqu'un un pair *A* reçoit un IHU d'un pair *B*, et *B* n'est pas un voisin symétrique, il supprime le cas échéant *B* de la liste de voisins potentiels ou unidirectionnels, et ajoute *B* à sa liste de voisins symétriques. Dans tous les cas, il met à jour la date du dernier *IHU* reçu.

Périodiquement, un pair parcourt ses listes de voisins unidirectionnels et symétriques et supprime les voisins qui ont expirés (la date du dernier paquet reçu est plus de 100 secondes dans le passé, ou la date du dernier IHU reçu est plus de 300 secondes dans le passé).

Enfin, toutes les quelques minutes, si le nombre de voisins potentiels est inférieur à 5, un pair envoie un TLV *Neighbour Request* à un voisin bidirectionnel choisi au hasard.

4.3 Publication

Un pair publie ses donées dans trois circonstances :

- au démarrage ;
- lorsque les données changent ;
- au moins une fois toutes les 30 minutes.

La publication consiste simplement à incrémenter le numéro de séquence et noter la date de dernière publication puis à l'inonder (voir paragraphe 4.5).

4.4 Réception de messages

Le comportement d'un pair lorsqu'il reçoit un TLV *IHU* est décrit à la partie 4.2.

Lorsqu'il reçoit un TLV *Neighbour Request*, un pair répond par un paquet contenant un TLV *Neigbours* contenant un sous ensemble tiré au hasard de ses voisins symétriques (au moins 5 voisins si possible).

Lorsqu'il reçoit un TLV *Neighbours*, un pair s'en sert pour repeupler sa liste de voisins potentiels.

Lorsqu'il reçoit un TLV *Data*, un pair vérifie si sa table de données contient déjà une donnée indexée par le même Id. Si c'est le cas, il compare les Seqno ; si le Seqno reçu est strictement supérieur, il met à jour la donnée, et lance une inondation, sinon il ne fait rien. Si la donnée n'est pas encore présente, il l'insère, met à jour la date de dernière publication, et lance une indondation. Dans tous les cas, il répond par un TLV *IHave* pour la donnée — il est *essentiel* de toujours envoyer un *IHave* pour chaque *Data* reçu.

Le TLV *IHave* est utilisé lors de l'inondation (voir ci-dessous). Lorsqu'un pair reçoit un TLV *IHave* correspondant à une donnée qui n'est pas en cours d'inondation, il l'ignore.

4.5 Inondation

Lorsque le numéro de séquence d'une donnée est incrémenté (soit parce que la donnée est publiée par le pair local, ou après la réception d'un TLV *Data*), un pair inonde cette donnée à tous ses voisins symétriques.

Pour cela, il construit une liste L de voisins qui n'ont pas encore acquitté la donnée, initialisée à la liste de voisins symétriques.

Toutes les trois secondes, il envoie à chaque membre de L la donnée. S'il reçoit d'un membre de L un TLV *Data* ou un TLV *IHave* contenant l'Id de la donnée inondée et un Seqno supérieur ou égal à celui de la donnée, il supprime l'émetteur du TLV de la liste L.

L'indondation termine normalement lorsque la liste L est vide. Si au bout de 11 secondes la liste L n'est pas vide, les pairs restant dans L sont supprimés de la liste de pairs symétriques (et il est bon dans ce cas de logger un message d'erreur).

4.6 Expiration de données obsolètes

Chaque donnée contient la date à laquelle le pair courant l'a apprise pour la première fois ; cette date n'est mise à jour que lorsque le numéro de séquence est incrémenté. Périodiquement, un pair parcourt sa table de données et supprime toutes les données qui ont été publiées plus de 35 minutes dans le passé.

5 Extensions

Interface utilisateur Le protocole ne définit bien sûr pas l'interface utilisateur. On peut imaginer une implémentation qui n'a pas d'interface utilisateur, qui publie une donnée statique (toujours la même) et pariticipe à l'inondation en arrière plan. On peut aussi imaginer une interface utilisateur raffinée, qui permet de visualiser les données inondées et publier de nouvelles données.

Ne passez pas trop de temps sur l'interface utilisateur — c'est un projet de réseaux, pas d'interfaces graphiques ou de programmation *web*.

IPv4 et IPv6 Le protocole est conçu pour ne pas dépendre de la famille d'adresses — Le seul endroit où les adresses IP apparaissent est le TLV *Neighbours*, et il supporte les deux familles d'adresses. Par contre, votre implémentation contiendra des adresses de socket dans les tables de

voisins. Votre implémentation pourra gérer les adresses IPv4, les adresses IPv6, ou les deux. Si vous gérez les deux familles d'adresses, il faudra gérer le cas d'un même pair qui est voisin deux fois.

Adresses et interfaces multiples La description du protocole ci-dessus suppose que votre pair n'a qu'une interface et qu'une adresse. En pratique, un pair peut avoir plusieurs interfaces, et plusieurs adresses sur la même interface. Pour gérer ce cas, il faudra que vos structures de données soient capables de gérer un pair qui est dans plusieurs relations de voisinage avec votre pair (une généralisation du problème du *double-stack* ci-dessus), et il faudra aussi prendre soin d'envoyer à un pair un IHU depuis l'adresse sur laquelle on reçoit ses paquets. Plusieurs solutions sont possibles.

Agrégation des messages La description du protocole suppose qu'un paquet ne contient qu'un seul TLV. Il est désirable d'agréger plusieurs TLV dans un seul paquet, et d'éviter d'envoyer des paquets vides lorsqu'on a déjà envoyé un paquet récemment. S'il est possible d'envoyer des paquets de 4096 octets (paragraphe 3), il est préférable pour des raisons d'efficacité d'éviter la fragmentation et donc de se limiter à 1460 environ lorsque c'est possible (ou le MTU moins 28 octets, si vous savez le déterminer).

La technique que je préfère consiste à gérer une file de messages en attente d'être envoyés, et d'envoyer un ou plusieurs paquets dès que la file est suffisamment longue ou que l'élément le plus ancien est suffisamment vieux (ce qui peut dépendre du type de TLV — il faut notamment prendre soin d'envoyer les TLV *IHave* en réponse à un *Data* de façon à ce qu'ils arrivent dans les trois secondes). Le cas du paquet vide peut être géré en ayant un type de paquet fictif, qui ne correspond à aucun TLV et ne sert qu'à maintenir une *deadline*.

Détection de voisins La détection de voisins décrite au paragraphs 4.2 est très naïve. Plusieurs améliorations sont possible qui améliorent ses performances sur des liens instables (Wifi etc.).

Accélération de la convergence Il y a plusieurs modifications mineures qui permettent d'accélérer la convergence. Par exemple, si un voisin nous envoie un paquet pour la première fois, il peut être intéressant de lui envoyer un *IHU* immédiatement. De même, si un voisin vient de nous envoyer un *IHU* pour la première fois, on peut lancer une inondation vers ce voisin (et ce voisin uniquement).

Inondation concurrente Dans la description ci-dessus, l'indondation interromp l'exécution du reste du protocole : pendant les 11 secondes (cas pire) de l'inondation, les autres TLV sont ignorés.

Une extension naturelle consiste à continuer la gestion du protocole pendant l'inondation, et même à inonder plusieurs TLV simultanément. On peut faire ça à l'aide de *threads* (ce que je vous déconseille), ou en intégrant l'inondation à la boucle principale (ce qui demande un peu de réflexion à propos des structures de données).

Inondation optimisée L'inondation décrite ci-dessus est très naïve : elle envoie systématiquement un TLV *Data* à chaque voisin, même celui qui nous a appris la nouvelle donnée. Plusieurs optimisations sont possibles, notamment :

- ne pas envoyer de TLV *Data* à un voisin qui nous a déjà envoyé un *Data* correspondant ;
- ne pas envoyer de paquet *Data* à un voisin qui a déjà envoyé un *IHave* correspondant ;
- retarder l'émission des TLV *Data* vers certains voisins pour diminuer la redondance de l'inondation.

Calcul de RTT L'inondation peut être optimisée de façon intéressante si on connaît le RTT qui nous sépare de chaque voisin. Malheureusement, le protocole est asynchrone : un pair peut retarder la réponse à un TLV de façon assez arbitraire, un algorithme naïf (à la *ping*) ne peut donc pas être utilisé.

Une bonne solution est d'utiliser l'algorithme de Mills, que nous avons vu lors du TP sur NTP. Il faudra ajouter trois *timestamps* au TLV *IHU*, ce qui peut être fait en ajoutant un sous-TLV au TLV *IHU*. Si vous définissez un tel TLV, coordonnez-vous avec moi pour éviter les collisions.

Découverte de voisins sur le lien local Le protocole découvre de nouveaux voisins de proche en proche, les apprenant d'un pair normalement distant. Pour améliorer la structure du graphe de voisinage, il peut être intéressant de découvrir des voisins locaux au lien.

Notez que ce n'est pas tout à fait évident. Comme les adresses locales au lien ne sont pas utilisables globalement, pour pouvoir annoncer les pairs locaux aux pairs distants il faudra définir un protocole qui permet d'apprendre une adresse globale après avoir découvert une adresse locale. L'alternative consiste a communiquer avec les pairs locaux, mais à ne pas les annoncer globalement.

Sécurité Ce protocole est vulnérable à toutes sortes d'attaques. La plus simple consiste à supprimer la donnée publiée par un pair en publiant une autre donnée avec un numéro de séquence supérieur.

Je serais heureux de voir des extensions au protocole qui évitent ce problème. Points en plus si vos extensions interopèrent avec le protocole de base, ce qui peut peut-être se faire en concaténant les paquets avec une signature cryptographique (c'est pour ça que les données supplémentaires dans le paquet sont ignorées), ou en signant les données publiées (ce qui demandera de définir un nouveau TLV contenu dans le TLV *Data*).

Autres extensions Toutes les autres extensions seront étudiées avec intérêt et bonne volonté. Sauf si elles sont vraiment trop débiles.

6 Modalités de soumission

Le projet sera fait par groupes de deux étudiants[1]. Vous devrez me remettre avant la soutenance :

1. $3 \neq 2$.

– le source complet de votre programme, accompagné d'un fichier `README` indiquant comment le compiler et s'en servir ;
– un rapport[2] sous format PDF contenant notamment une description des extensions que vous avec implémentées.

Vous pouvez choisir le langage de programmation que vous voulez, du moment que je pourrai facilement tester votre projet sous *Debian Stable*.

Vous me soumettrez votre solution par courrier électronique à mon adresse. Le courrier que vous m'enverrez devra impérativement avoir le sujet « `Projet reseaux Cachan` » avec une archive `.tar.gz` en attachement. L'archive devra s'appeler *nom1*`-`*nom2*`.tar.gz`, et s'extraire dans un sous-répertoire *nom1*`-`*nom2* du répertoire courant. Par exemple, si vous vous appelez *Boris Johnson* et *Donald Trump*, votre archive devra porter le nom `johnson-trump.tar.gz` et son extraction devra créer un répertoire `johnson-trump` contenant tous les fichiers que vous me soumettrez.

2. La concision est une vertu.