Simulation du protocole de routage RIP

Pierre Lecavelier - Adrien Laurence

22 décembre 2007

Table des matières

1 Présentation de l'application RipSim

Dans cette partie, nous allons présenter l'application créée pour simuler le protocole de routage RIP.

Même si nous avons fait en sorte que cette application soit la plus intuitive possible, il convient tout de même d'en présenter les principales fonctionnalités.

RipSim permet de créer un inter-réseau composé de machines et de réseaux, puis de les connecter entre eux. On peut bien sur activer RIP pour déclencher la mise à jour des tables de routage, par propagation de celles-ci entre les machines.

1.1 Les inter-réseaux

RipSim permet donc la gestion d'inter-réseaux. Voici une capture d'écran présentant cette application.

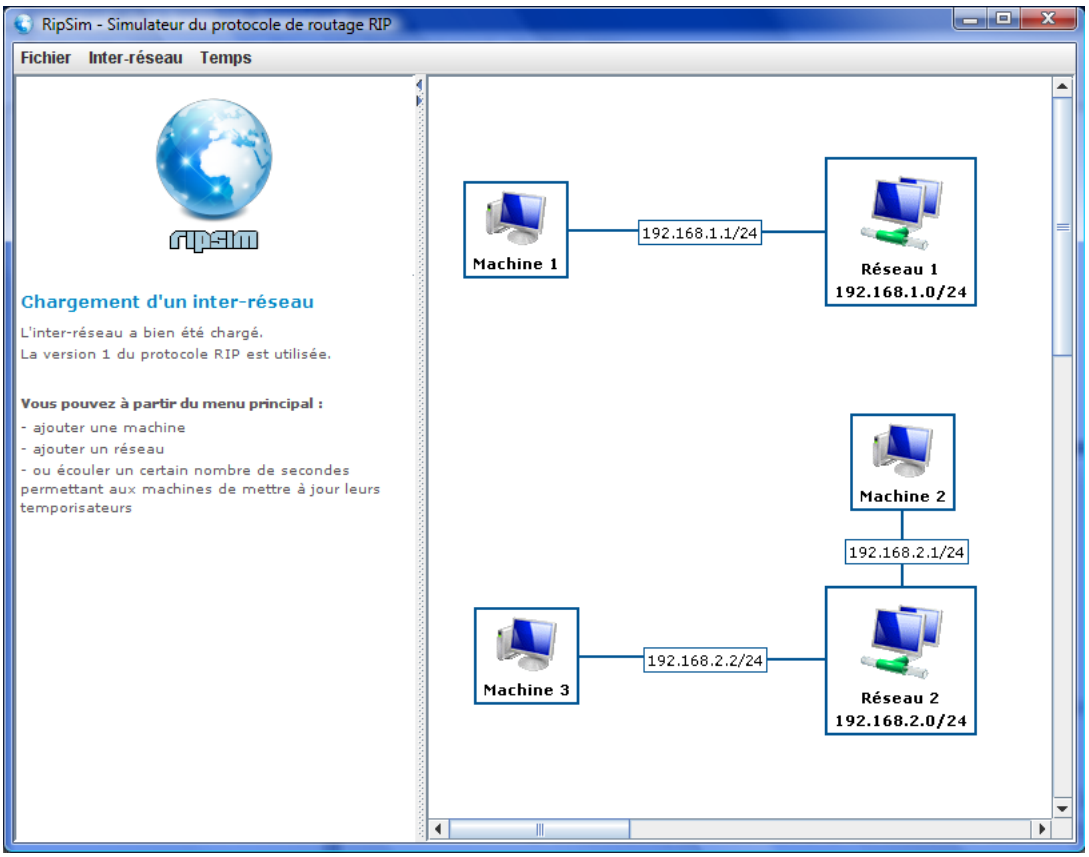

FIG. 1 – Application RipSim

1.1.1 Création, ouverture et sauvegarde d'un inter-réseau

Dans la capture 2, on peut voir que l'on peut créer un nouvel inter-réseau. Les deux version de RIP ont été implémentées ; nous verrons au fur et à mesure de ce document les différences entres ces versions.

On peut également en ouvrir un existant. Les fichiers supportés par RipSim portent l'extension *.rip*.

Et enfin, on peut sauvegarder le réseau courant.

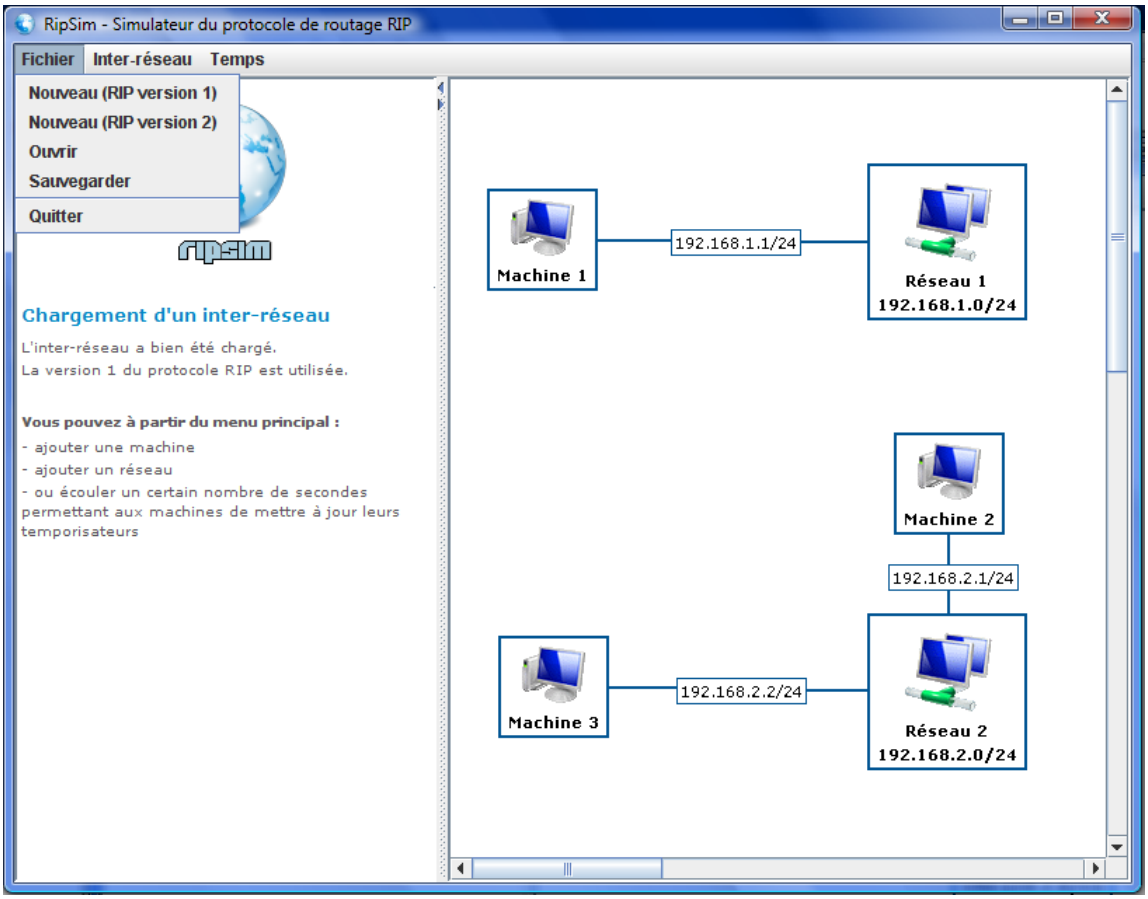

FIG. 2 – Gestion des inter-réseaux

1.1.2 Construction de l'inter-réseau

Une fois qu'un inter-réseau a été ouvert dans la fenêtre, il faut désormais pouvoir ajouter des machines et des réseaux. C'est ce que permet le menu *Inter-réseau*, comme on peut le voir sur la capture 3.

Remarque : La création d'une machine ne différencie pas les passerelles et les hôtes. Tout dépend à combien de réseaux est connectée la machine.

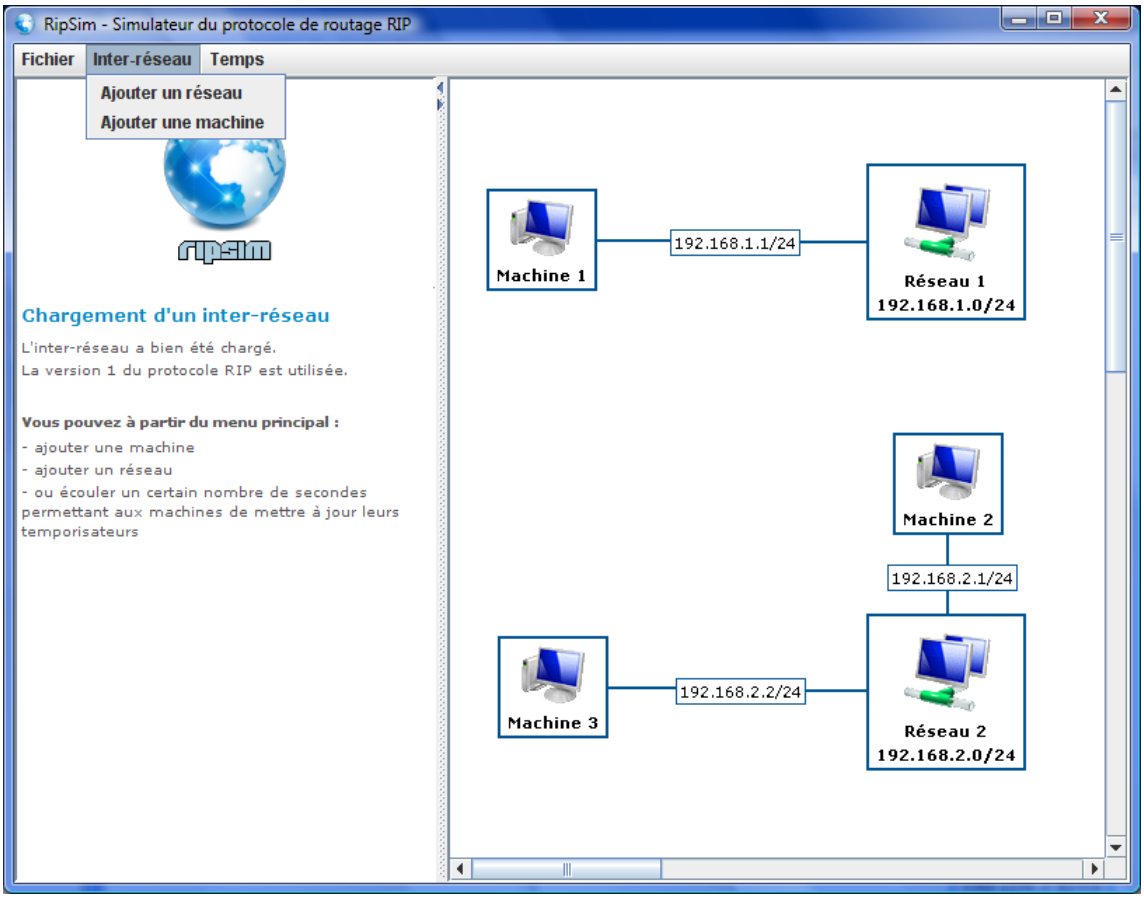

FIG. 3 – Gestion des inter-réseaux

Pour créer un réseau, il faut définir son adresse IP pour les versions 1 et 2 de RIP puis son masque seulement pour la version 2 (capture 4).

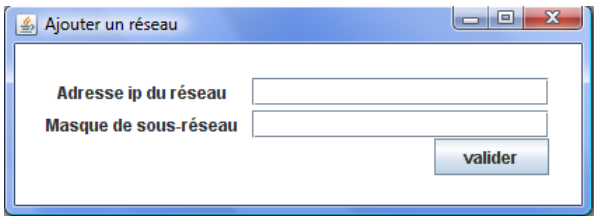

FIG. 4 – Ajout d'un réseau

1.1.3 Gestion du temps

Comme on le verra plus loin dans ce document, le protocole RIP fonctionne avec des temporisateurs. Pour éviter de mettre en place un fonctionnement en "temps réel" ce qui aurait grandement complexifier l'application (notamment par la mise en place de threads), nous avons préféré donner la possibilité d'écouler un certain nombre de secondes (capture 5). Ainsi, toutes les 30 secondes, chaque machine fonctionnant avec RIP, mettra à jour ces temporisateurs et déclenchera les actions correspondantes.

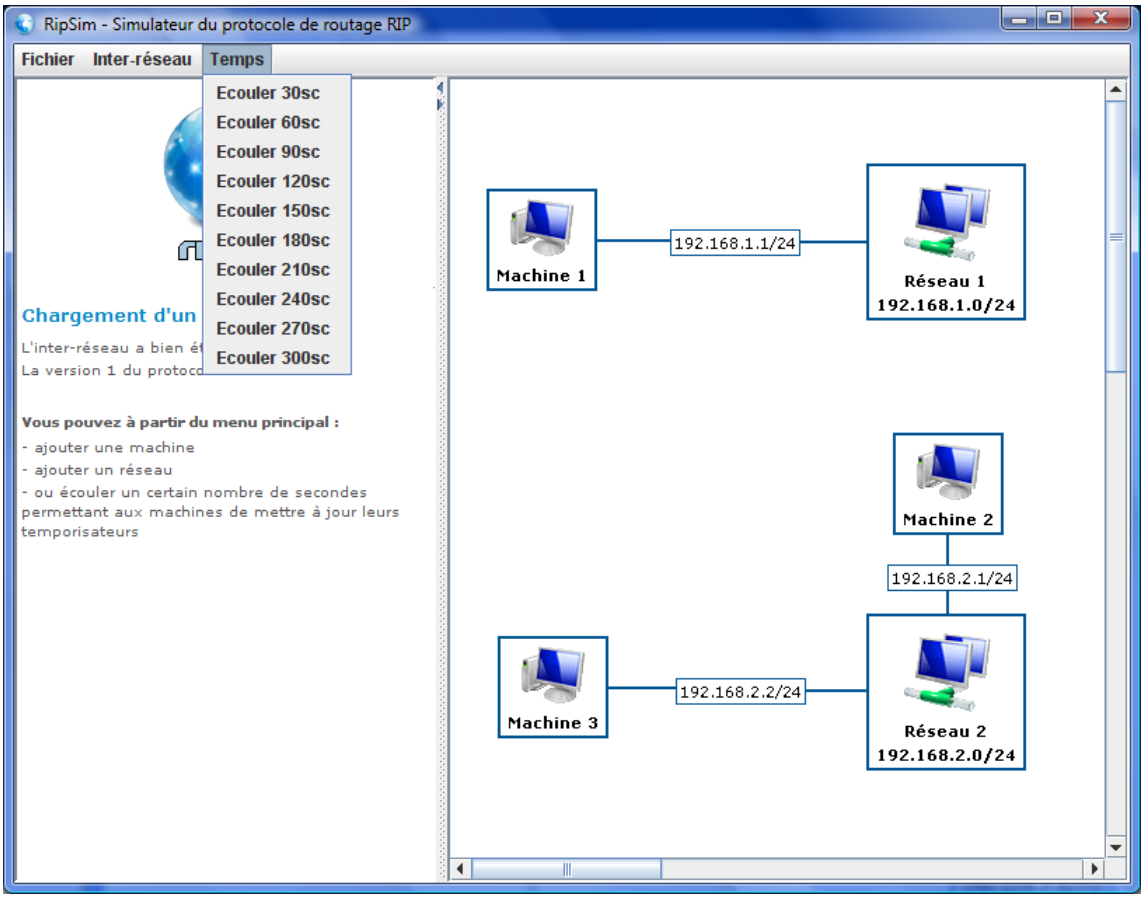

FIG. 5 – Gestion du temps

1.2 Les réseaux

Une fois qu'un réseau a été placé sur l'inter-réseau, vous pouvez cliquer dessus pour afficher certaines informations le concernant, notamment les machines connectées, la classe du réseau, son adresse ip/masque mais aussi pour savoir si il est public ou privé. Un clic droit ouvrira une fenêtre popup permettant d'agir sur ce réseau (capture 6).

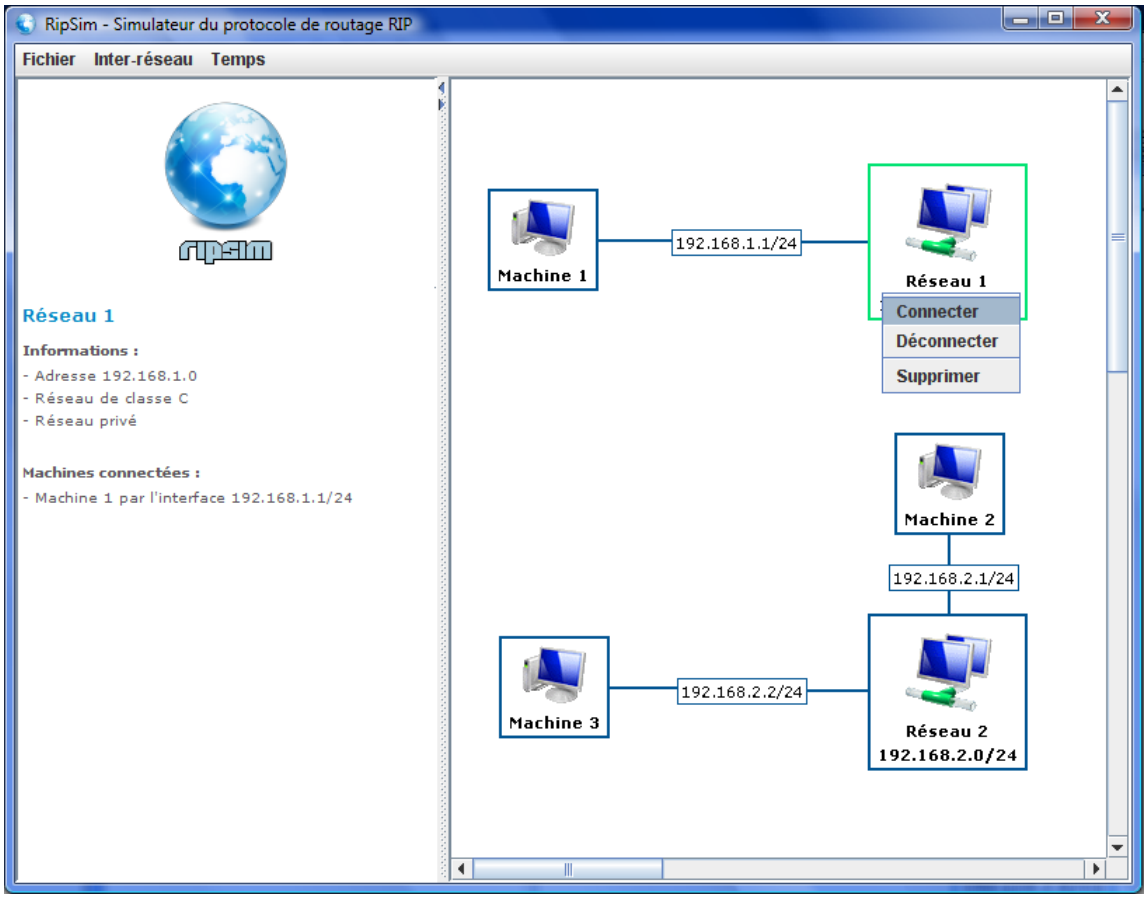

FIG. 6 – Les réseaux

1.2.1 Connexion/déconnexion

Ces deux fonctionnalités permettent donc de connecter ou déconnecter ce réseau d'une machine (captures 7 et 8).

Remarque : L'application ne permettant pas les liaisons point à point, vous pouvez seulement utiliser la connexion/déconnexion d'un réseau à une machine (et non pas entre deux machines).

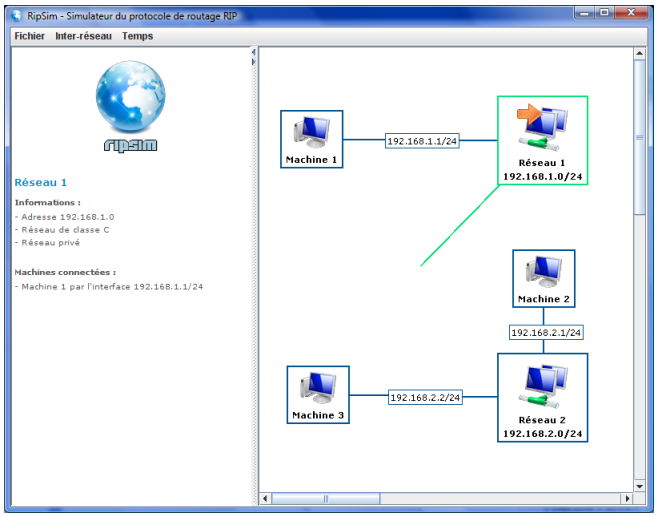

FIG. 7 – Choix de la machine

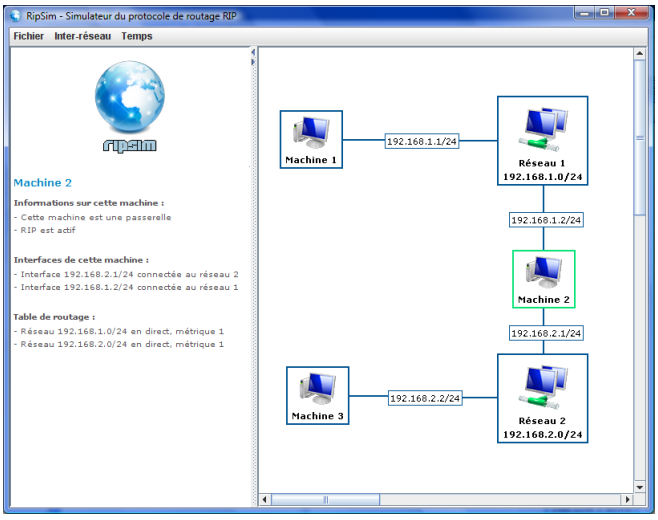

FIG. 8 – Connexion à la machine

1.2.2 Suppression

Cette fonction permet tout simplement la suppression du réseau.

1.3 Les machines

Une fois qu'une machine a été placé sur l'inter-réseau, vous pouvez cliquer dessus pour afficher certaines informations la concernant, notamment ses interfaces et sa table de routage. Un clic droit ouvrira une fenêtre popup permettant d'agir sur cette machine (capture 9).

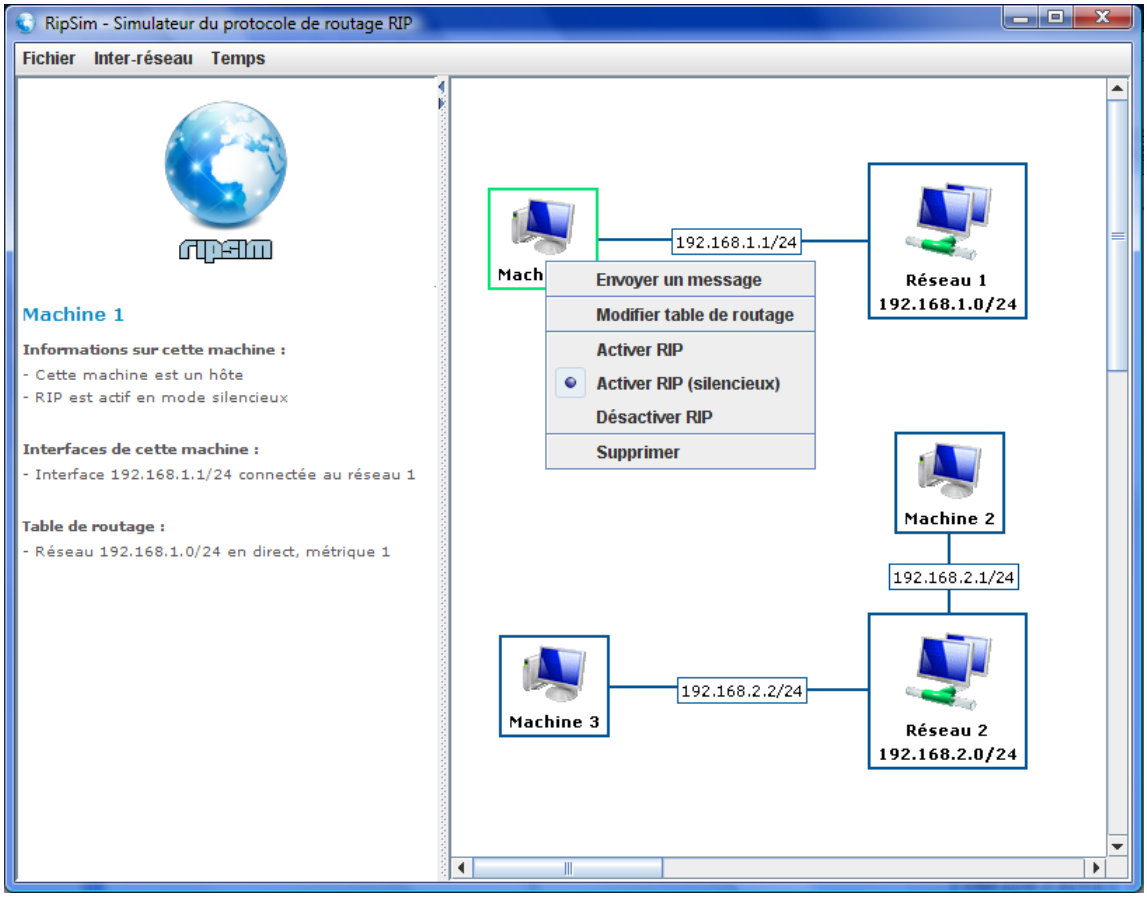

FIG. 9 – Les machines

1.3.1 Envoi d'un message

Cette fonctionnalité permet donc d'envoyer un datagramme contenant un message à une autre machine (captures 10 et 11). Elle permet de visualiser les routes choisies pour le routage du datagramme et donc de vérifier le bon fonctionnement du protocole RIP qui vise à choisir la route la plus courte pour atteindre une destination.

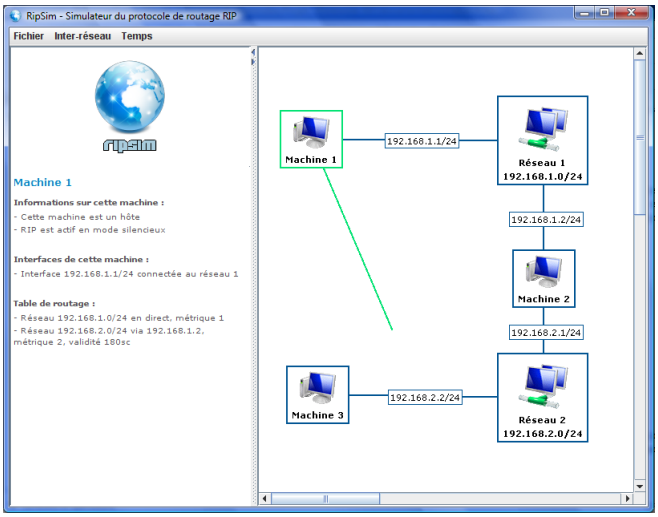

Fig. 10 – Choix du destinataire

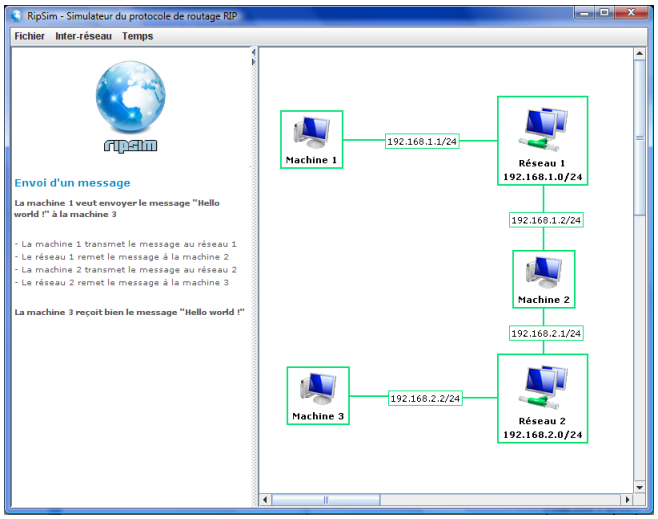

Fig. 11 – Envoi du message

1.3.2 Modification de la table de routage

Vous pouvez voir sur la capture 12 la fenêtre pour modifier la table de routage.

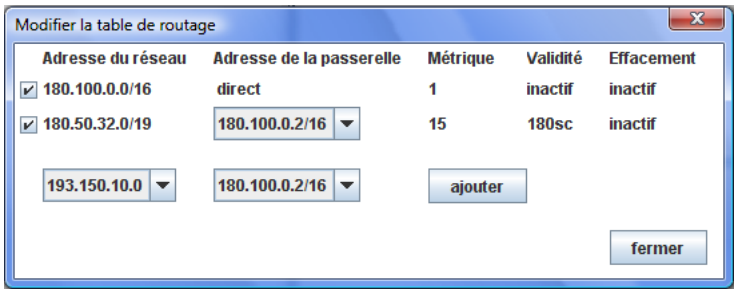

Fig. 12 – Modification de la table de routage

3 actions sont possible à partir de cette fenêtre :

- L'activation ou la désactivation d'une route grâce aux checkbox. La désactivation aura pour effet d'entamer un processus d'effacement de la route (métrique à 16 et temporisateur de ramassage des déchets initialisé). Alors que l'activation revalidera une route auparavant inactive (en fixant la métrique à 15 ou à 1 pour un réseau directement connecté et en supprimant le temporisateur de ramassage de déchets).
- Le changement de passerelle modifiera la route vers le réseau concerné. A chaque changement de passerelle, on fixe en plus la métrique à 15.
- L'ajout d'une route ajoutera une ligne à la table de routage vers le réseau concerné en passant par la passerelle choisie. La métrique sera fixée à 15 ou à 1 pour un réseau directement connecté.

Comme vous pouvez le constater, on fixe la métrique à 15 pour l'activation, le changement de passerelle et l'ajout d'une route dans les cas ou le réseau n'est pas directement connecté. Nous avons choisie cette valeur car dans ces 3 cas, nous ne connaissons pas la métrique associée à la route. Donc, le fait de prendre la valeur juste en dessous de l'infinie 16 aura pour effet de toujours remplacée cette route par une nouvelle dont on connait la métrique après la réception d'un datagramme de mise à jour RIP.

1.3.3 Modification du protocole RIP

Cette option permet de modifier le protocole RIP sur la machine. On peut alors le désactiver, l'activer en mode silencieux (les messages de mises à jour seront pris en compte mais pas diffusés) ou en mode passerelle (les messages de mises à jour seront pris en compte et diffusés).

1.3.4 Suppression

Cette fonction permet tout simplement de retirer la machine de l'inter-réseau.

2 Structures de données et modélisation

2.1 Diagramme UML

Voici un datagramme UML (Fig. 13) représentant les différents packages intervenant dans la partie réseau de l'application.

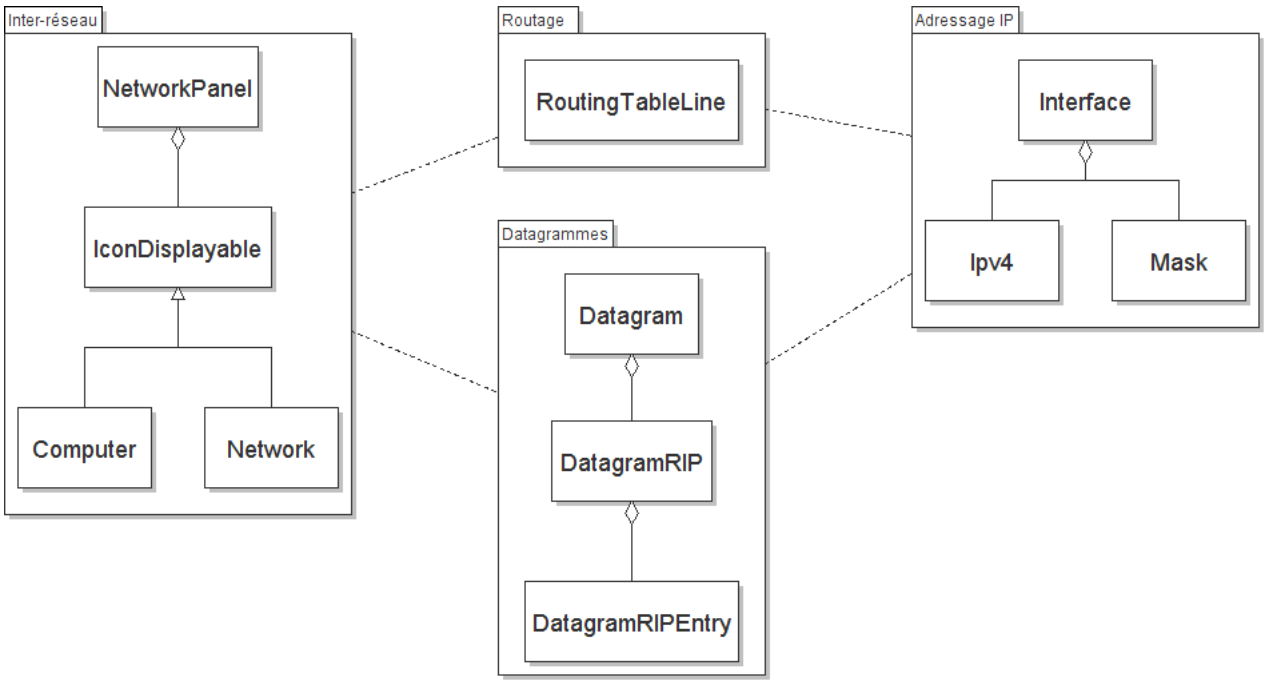

FIG. 13 – Diagramme UML de l'application RipSim

Remarque : Toutes les autres classes non décrites dans ce document permettent de déclencher les actions et de gérer l'affichage.

Tout d'abord, le package *Inter-réseau* permet de représenter un inter-réseau composé de réseaux et de machines.

Remarque : La classe *IconDisplayable* n'intervient à aucun moment dans la partie réseau de l'application. Elle sert simplement à afficher un icône sur le schéma de l'inter-réseau.

Ensuite, le package *Inter-réseau* utilise les packages *Datagrammes* et *Routage*. Le premier permet de représenter un datagramme IP qui peut contenir un datagram RIP lui-même contenant différentes entrées. Le deuxième contient une seule classe représentant une ligne de la table de routage.

Enfin, les 3 packages précédents utilisent le package *Adressage IP* permettant de gérer les interfaces d'une machine, les adresses IP (version 4) et les masques.

Dans les parties suivantes, nous allons détaillé chacun des packages et leurs classes. Pour ne pas alourdir ce rapport, nous allons essentiellement nous concentrer sur les attributs et les méthodes intervenant dans la partie réseau de cette application (connexions/déconnexions, routage des datagrammes, mises à jour des tables de routage, etc...)

2.2 Inter-réseau

2.2.1 Classe NetworkPanel

Cette classe est utilisée pour représenter et afficher l'inter-réseau dans le panneau de droite. Elle sert donc à stocker l'ensemble des machines et des réseaux.

Attributs :
- *int ripVersion*
  Version de RIP utilisée (1 ou 2) pour l'inter-réseau.
- *List<Network> networks*
  Liste pour stocker tous les réseaux.
- *List<Computer> computers*
  Ensemble des machines.

Méthodes :
- *NetworkPanel(NetworkSimulator simulator, int ripVersion)*
  Constructeur de la classe qui prend le contrôleur de l'application et la version du protocole RIP utilisée en arguments.
- *update() : void*
  Cette méthode met à jour les timers de toutes les machines de l'inter-réseau.

2.2.2 Classe Network

Cette classe représente un réseau et permet la connexion/déconnexion à une machine mais aussi la diffusion d'un datagramme à une ou plusieurs machines connectées.

Attributs :
- *Ipv4 ip*
  Adresse ip du réseau.
- *Mask mask*
  Masque de sous-réseau.
- *int cost*
  coût du réseau (égal à 1 pour notre application).
- *List<Computer> computers*
  Liste des machines connectées.

Méthodes :
- *Network(NetworkSimulator simulator, Ipv4 ip, Mask mask)*
  Constructeur de la classe qui prend le contrôleur de l'application, une adresse IP et un masque en arguments.
- *connect(Computer cpu) : Boolean*
  Méthode qui connecte la machine passée en argument en lui assignant une adresse IP. Si le réseau n'a plus d'IP disponible, elle retourne *faux*.
- *deconnect(Computer cpu) : void*
  Méthode pour déconnecter la machine passée en argument.
- *assignIp() : Ipv4*
  Méthode qui retourne une adresse IP disponible.
- *receiptDatagram(Datagram datagram, Interface senderInterface, Ipv4 receipterAdress) : void*
  Méthode pour recevoir un datagramme provenant de l'interface d'une machine connectée, qui sera ensuite diffusé à l'adresse de réception passée en argument.
- *sendDatagram(Datagram datagram, Ipv4 receipterAdress) : void*
  Méthode envoyant un datagramme à l'adresse de réception passée en argument. Si cette adresse correspond à l'adresse de broadcast du réseau, le datagramme sera diffusé à toutes les machines connectées, sinon il sera envoyé à la machine concernée.

2.2.3 Classe Computer

Cette classe permet de représenter indifféremment une machine hôte ou passerelle. C'est dans cette classe qu'est implémenté le protocole RIP (pour plus de détails, veuillez consulter la *partie 3 - Implémentation du protocole*).

Attributs :
- *List<Network> links*
  Liste de réseaux connectés à la machine.
- *List<Interface> interfaces*
  Liste des interfaces.
- *List<RoutingTableLine> routingTable*
  Table de routage.
- *String ripMode*
  Mode du protocole RIP utilisé (inactif, actif ou actif silencieux).
- *String ripPassword*
  Mot de passe pour identifier les datagrammes RIP reçus (dans notre cas, il vaudra "password" pour toutes les machines).

Méthodes :
- *Computer(NetworkSimulator simulator)*
  Constructeur de la classe qui prend le contrôleur de l'application en argument.
- *connect(Network network, Interface interf) : void*
  Méthode pour connecter la machine au réseau par l'interface passée en argument.
- *deconnect(Network network) : void*
  Méthode qui déconnecte la machine du réseau passé en argument.
- *sendDatagram(Datagram datagram) : void*
  Méthode qui envoie un datagramme.
- *receiptDatagram(Datagram datagram, Interface interf) : void*
  Méthode pour recevoir un datagramme par l'interface passée en argument.
- *isReceipter(Datagram datagram, Interface interf) : Boolean*
  Méthode vérifiant si la machine est bien le destinataire du datagramme.
- *sendMessage(String message, Ipv4 receipterAdress) : void*
  Méthode pour envoyer un message. La machine va donc créer puis envoyer un datagramme (méthode *sendDatagram*) avec le protocole de transport TCP et qui contiendra le message passé en argument.
- *receiptMessage(Datagram datagram, Interface interf) : void*
  Méthode pour recevoir un message.
- *updateWithRIP() : void*
  Méthode pour déclencher une mise à jour des tables de routage au sein de l'inter-réseau.
- *createRIPDatagram(Network network) : DatagramRIP*
  Méthode qui crée un datagramme RIP à destination du réseau passé en argument.
- *receiptRIPDatagram(Datagram datagram, Interface interf) : void*
  Méthode pour recevoir un datagramme contenant un datagramme RIP par l'interface passée en argument. La machine mettra à jour sa table de routage grâce à ce datagramme.
- *updateTimers() : void*
  La machine mettra à jour ses temporisateurs (de validité ou de ramassage de déchets selon le cas) avec cette méthode.

2.3 Adressage IP

2.3.1 Classe Interface

Cette classe est utilisée pour représenter l'interface d'une machine.

Attributs :
- *Ipv4 ip*
  Adresse IP de l'interface.
- *Mask mask*
  Masque utilisé pour l'interface.

Méthodes :
- *Interface(Ipv4 ip, Mask mask)*
  Constructeur de la classe qui prend une adresse IP et un masque en arguments.

2.3.2 Classe Ipv4

Classe pour gérer l'adressage IP (version 4).

Attributs :
- *String binary*
  Notation binaire de l'adresse.
- *String networkClass*
  Classe de l'adresse.
- *Mask classNetworkMask*
  Masque du réseau correspondant à l'adresse.
- *String access*
  Accès de l'adresse (privée ou publique)

Méthodes :
- *Ipv4(String ip)*
  Constructeur de la classe qui prend une adresse IP sous la forme w.x.y.z, la vérifie, l'analyse et la convertit en binaire.
- *analyse() : void*
  Méthode pour analyser l'adresse (plus exactement pour définir sa classe et son accès).
- *getNetworkIp(Mask mask) : Ipv4*
  Méthode pour récupérer l'adresse du sous-réseau (tous les bits de l'host-id à 0) correspondant au masque de sous-réseau passé en argument.
- *getNetworkIp() : Ipv4*
  Méthode pour récupérer l'adresse du réseau correspondant à l'adresse.
- *getBroadcastIp(Mask mask) : Ipv4*
  Méthode pour récupérer l'adresse de diffusion du sous-réseau (tous les bits de l'host-id à 1) correspondant au masque de sous-réseau passé en argument.
- *getNetworkIp() : Ipv4*
  Méthode pour récupérer l'adresse de diffusion du réseau correspondant à l'adresse.
- *getNetId(Mask mask) : String*
  Méthode pour récupérer le net-id de l'adresse avec le masque passé en argument.
- *getHostId(Mask mask) : String*
  Méthode pour récupérer l'host-id de l'adresse avec le masque passé en argument.

2.3.3 Classe Mask

Classe représentant un masque.

Attributs :
- *int cidr*
  Notation CIDR du masque.
- *String binary*
  Notation binaire du masque.

Méthodes :
- *Mask(int cidr)*
  Constructeur de la classe qui prend une notation CIDR sous forme décimale en argument.
- *Mask(String cidr)*
  Constructeur de la classe qui prend une notation CIDR sous forme de chaîne en argument, et qui la vérifie.

2.4 Table de routage

2.4.1 Classe RoutingTableLine

Cette classe permet de représenter une ligne de la table de routage. Donc, comme nous l'avons vu plus haut, la classe *Computer* possède une liste de lignes pour définir sa table de routage.

Attributs :
- *Ipv4 networkIp*
  Adresse de réseau ou de sous-réseau.
- *Mask mask*
  Masque de réseau ou de sous-réseau.
- *Ipv4 interfaceIp*
  Adresse de l'interface pour envoyer un datagrame.
- *Ipv4 gatewayIp*
  Adresse de la passerelle pour router un datagramme.
- *int metric*
  Métrique de la route.
- *int timer*
  Temporisateur de validité.
- *int trashTimer*
  Temporisateur de ramassage des déchets.
- *Boolean flag*
  Drapeau de changement de route.

Méthodes :
- *RoutingTableLine(Ipv4 networkIp, Mask mask, Ipv4 interfaceIp, Ipv4 gatewayIp, int metric, Boolean flag, Integer timer, Integer trashTimer)*
  Constructeur de la classe.

2.5 Datagrammes

2.5.1 Classe Datagram

Cette classe permet de construire un datagramme IP. Comme indiqué dans le rapport, nous en avons simplifié la structure. Soit le datagramme contient un message dans sa zone de données, le protocole de transport utilisé dans ce cas sera TCP. Soit il contient un datagramme RIP, et le protocole de transport sera alors UDP.

Attributs :
- *Ipv4 senderAdress*
  Adresse de l'émetteur.
- *Ipv4 senderAdress*
  Adresse du destinataire.
- *int ttl*
  Time to leave fixé à 50.
- *int protocol*
  Protocole utilisé pour le transport.
- *String message*
  Message à envoyer.
- *DatagramRIP rip*
  Datagramme RIP à diffuser.

Méthodes
- *Datagram(Ipv4 senderAdress, Ipv4 receipterAdress, String data)*
  Constructeur pour créer un datagramme contenant un message (variable *protocol* fixé à 6 pour TCP).
- *Datagram(Ipv4 senderAdress, Ipv4 receipterAdress, String data)*
  Constructeur pour créer un datagramme contenant un datagramme RIP (variable *protocol* fixé à 17 pour UDP).

2.5.2 Classe DatagramRIP

Cette classe permet de créer un datagramme RIP. On utilisera seulement les messages de type "response", car notre application ne permet que des mises à jour déclenchées suite à la vérification des temporisateurs.

Attributs :
- *int command*
  Commande utilisée pour le datagramme (égale à 2 pour un message de type "response").
- *int version*
  Version du protocole utilisée.
- *List<DatagramRIPEntry> entries*
  Entrées du datagramme.

Méthodes
- *DatagramRIP(int version)*
  Constructeur de la classe.

2.5.3 Classe DatagramRIPEntry

Cette classe représente une entrée d'un datagramme RIP. On peut avoir 3 types d'entrée différents, une entrée pour la version 1 du protocole avec adresse IP et métrique, une entrée pour la version 2 avec le masque en plus, et enfin une entrée d'identification à nouveau pour la version 2 du protocole RIP.

Attributs :
- *String family*
  Identificateur de famille d'adresses.
- *Ipv4 adress*
  Adresse du réseau de destination.
- *int metric*
  Métrique.
- *Mask mask*
  Masque de sous-réseau.
- *int authorType*
  Type d'identification (fixé à 2 pour un simple mot de passe).
- *String password*
  Mot de passe utilisé.

Méthodes
- *DatagramRIPEntry(Ipv4 adress, int metric)*
  Constructeur d'une entrée pour la version 1 du protocole.
- *DatagramRIPEntry(Ipv4 adress, Mask mask, int metric)*
  Constructeur d'une entrée pour la version 2 du protocole.
- *DatagramRIPEntry(String password)*
  Constructeur de l'entrée d'identification pour la version 2 du protocole.

3 Implémentation du protocole RIP

Dans cette partie, nous allons décrire l'implémentation du protocole RIP choisie pour notre programme. Tous les éléments présentés ci-dessous sont directement issus des spécifications du protocole. Nous ferons également référence aux parties de code à consulter pour en vérifier la bonne implémentation (le code est dûment commenté).

3.1 Spécifications des réseaux

Dans un premier temps, il faut définir un coût pour les réseaux. D'après le protocole *"Dans les implémentations RIP existantes, 1 est toujours utilisé comme coût."* Nous utiliserons également 1 comme coût pour chaque réseau.

De plus, en se basant une nouvelle fois sur le protocole *"Les entrées pour les réseaux directement connectés sont définies par l'hôte, en utilisant des informations récoltées par des moyens non spécifiés par ce protocole."* On en déduit qu'il faut mettre en place un système permettant de gérer les routes pour les réseaux directement connectés.

C'est ce système que nous allons présenter dans les 2 sections suivantes.

3.1.1 Connexion

Au moment de connecter une machine à un réseau, nous allons effectué les opérations suivantes : Si la route vers ce réseau existe déjà, on la remplace car étant donnée qu'elle est directe, elle sera forcément plus courte ou égale à l'ancienne. Sinon, on ajoute une nouvelle route directe vers ce réseau avec une métrique de 1.

Code : Vous pouvez consulter la méthode *connect(Network network, Interface cpuInterface)* de la classe *Computer*.

3.1.2 Déconnexion

Après la connexion, il faut bien sûr pouvoir gérer la déconnexion. Donc au moment de déconnecter une machine d'un réseau, on entame le processus d'effacement de la route vers ce réseau. Nous décrirons ce processus dans la partie *Temporisateurs*.

Code : Vous pouvez consulter la méthode *deconnect(Network network)* de la classe *Computer*.

3.1.3 Réseaux publics et privés

Comme demandé dans l'énoncé, nous avons pris en compte si un réseau est public ou privé.

Au moment d'envoyer un datagramme, on vérifie les adresses d'émission et de réception. Si au moins une des deux adresses est privée et que le réseau utilisé pour diffuser le datagramme est publique, on en annule l'envoi.

De plus, au moment de créer un message RIP, si le réseau de réception est public, alors on filtre seulement les lignes de la table de routage contenant une adresse publique.

Code : Vous pouvez consulter la méthode *sendDatagram(Datagram datagram)* de la classe *Computer* pour la vérification d'un datagramme.
Code : Vous pouvez consulter la méthode *createRIPDatagram(Network network)* de la classe *Computer* pour le filtrage des entrées d'un datagramme RIP.

Code : Vous pouvez consulter la méthode *analyse()* de la classe *Ipv4* qui définit si une adresse est publique ou privée.

3.1.4 Les pannes

Même si notre application ne le fait pas, ce serait essentiel de pouvoir gérer les pannes. Car que ce passerait-il si le réseau, l'interface d'une machine ou la connexion entre un réseau et une machine tombaient en panne ? La route resterait valide dans la table de routage alors qu'elle ne l'est plus.

L'idéal serait de rester à l'écoute des datagrammes envoyés. Si les acquittements de plusieurs datagrammes ne sont pas reçus, on pourrait considérer qu'il y a une panne. Dans ce cas, on entamerait le processus d'effacement.

3.2 Routage d'un datagramme

Dans cette partie nous allons décrire le routage d'un datagramme.

Au moment d'envoyer un datagramme, la machine émettrice va parcourir les lignes de sa table de routage. Si elle trouve une route vers le réseau du destinataire, elle vérifie si la connexion est directe et dans ce cas lui envoie directement le datagramme en passant par le réseau concerné. Au contraire, si la route est indirecte, elle envoie alors le datagramme à la passerelle indiquée dans la table de routage, toujours en passant par le réseau concerné.

Après plusieurs itérations, le datagramme arrivera finalement à destination (enfin si les tables de routage sont bien à jour).

Code : Vous pouvez consulter la méthode *sendDatagram(Datagram datagram)* de la classe *Computer*.

3.3 Temporisateurs

Cette section décrit la gestion des temporisateurs pour notre application.

Code : Vous pouvez consulter la méthode *updateTimers()* de la classe *Computer*.

3.3.1 Temporisateurs de la table de routage

Comme spécifié dans le protocole, deux temporisateurs sont utilisés pour chaque ligne de la table de routage :

- Un temporisateur de validité
- Un temporisateur de ramassage des déchets

Le premier est initialisé à 180 secondes. Tant qu'il est actif, cela indique que la route est bien valide. Par contre, une fois écoulé, le processus d'effacement est entamé.

Le processus d'effacement sert donc à supprimer une ligne de la table de routage. Mais pourquoi patienter avant de supprimer une route ? Tout simplement pour laisser le temps à la machine de propager l'information. Autrement dit, pour avoir le temps d'informer les autres machines que la route n'est plus valide.

Voici en quoi consiste notre processus d'effacement :

- On fixe la métrique à 16 (infinie) pour indiquer que la route n'est plus valide
- On désactive le temporisateur de validité
- On initialise le temporisateur de ramassage des déchets à 120 secondes

Une fois les 120 secondes dépassées, on supprime définitivement la ligne de la table de routage.

3.3.2 Temporisateur d'une machine

En plus des temporisateurs relatifs à la table de routage, nous avons mis en place un temporisateur pour chaque machine. Le but est de déclencher une mise à jour avec RIP toutes les 30 secondes.

Nous allons décrire dans la partie suivante les mises à jour déclenchées.

3.4 Mises à jour déclenchées

Nous allons détaillé dans cette partie comment notre application gère les mises à jour déclenchées toutes les 30 secondes par les machines.

Pour résumer, si RIP est actif sur une machine, cette dernière devra envoyer sa table de routage à tous ses voisins. A la réception d'un message RIP, chaque voisin mettra à jour et enverra à son tour sa propre table de routage. Toutes les machines devraient donc avoir leur table à jour après un certain laps de temps.

3.4.1 Table de routage d'une machine

Voici un récpitulatif des éléments composant chaque ligne d'une table de routage :

- Un adresse de réseau (ou sous-réseau)
- Un masque de réseau (ou sous-réseau)
- Une interface pour émettre un datagramme
- Une adresse de passerelle à qui envoyer un datagramme
- Une métrique correspondant au nombre de passerelles à traverser pour atteindre le réseau concerné
- Un drapeau de changement de route qui indique si la route a changé récemment
- Un temporisateur de validité de la route
- Un temporisateur de rammassage des déchets

Code : Vous pouvez consulter la classe *RoutingTableLine*.

3.4.2 Format des messages

On peut voir ci-dessous (Fig. 14) le format des messages envoyé par les machines fonctionnant avec RIP.

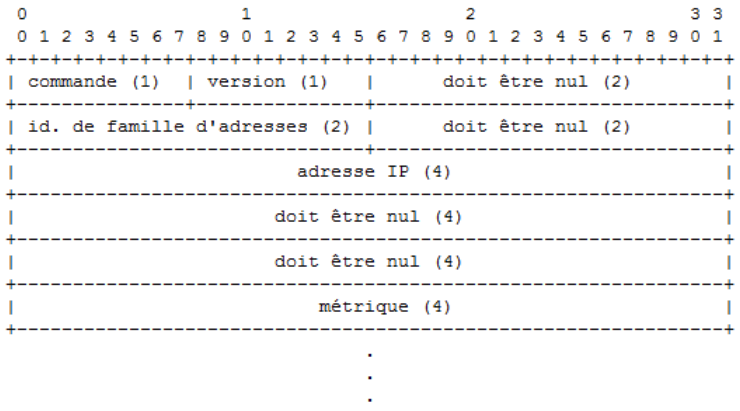

Fig. 14 – Format d'un message RIP version 1

L'en-tête contient la version de RIP et le type de commande utilisé. Dans notre cas, l'application ne gérant que les mises à jour déclenchées par un temporisateur, seul le type de commande *response* sera utilisé.

Ensuite le message contient un ensemble d'entrées composées de l'adresse du réseau de destination, de sa métrique associée et d'un identificateur de famille d'adresse qui sera fixé à 2 (pour les adresses internet).

Code : Vous pouvez consulter les classes *DatagramRIP* et *DatagramRIPEntry*.

3.4.3 Traitement de la sortie

Nous allons maintenant décrire le comportement d'une machine quand son temporisateur lui ordonne de déclencher une mise à jour.

Elle va d'abord créer le message contenant sa table de routage. Pour ce faire, elle va compléter ce message en y ajoutant une à une les lignes de sa table de routage (adresse du réseau de destination et métrique associée). A une exception près, si la passerelle de la route est située sur le réseau pour lequel le message est préparé, alors la métrique sera fixé à 16 dans l'entrée du message. Cela consiste à faire de l'horizon partagé avec empoisonnement (se référer au protocole pour plus de détails).

Finalement, la machine va envoyer ce message à tous ses voisins.

Code : Vous pouvez consulter les méthodes *updateWithRIP()* et *createRIPDatagram(Network network)* de la classe *Computer*.

3.4.4 Traitement de l'entrée

Une fois qu'un message RIP a été reçu par une machine, que fait notre application ? C'est ce que nous allons voir dans cette section.

Dans un premier temps, si la machine de réception du message RIP est à l'origine de ce même message, on ne fait rien.

Sinon elle va analyser le message reçu. Pour cela, elle va en parcourir chaque entrée et fixer la nouvelle métrique à utiliser :

$$nouvelle\ métrique = min(métrique\ de\ l'entrée + coût\ du\ réseau, 16)$$

Ensuite, si une route existe déjà dans la table de routage et que cette route est en connexion directe avec le réseau, on vérifie que cet accès direct est toujours valide (métrique différente de 16). Si ne n'est pas le cas, on remplace cet accès par la nouvelle route.

Sinon si une route existe déjà mais que la connexion est indirecte, on compare d'abord les passerelles. Si le message provient de la même passerelle que la route existante, on réinitialise la temporisation.

Puis on compare les métriques. Si le message provient de la même passerelle que la route existante et que la nouvelle métrique est différente de l'ancienne, ou si la nouvelle métrique est plus petite que l'ancienne, on fait les choses suivantes :

- On change la passerelle (machine émettrice du message) et l'interface de la machine.
- On prend la nouvelle métrique
- On positionne le drapeau de changement de route
- Puis on regarde la nouvelle métrique, si elle égale à 16 on entame le processus d'effacement (en initialisant le temporisateur de ramassage des déchets à 120 secondes), sinon on réinitialise le temporisateur de validité à 180 secondes

Par contre, si la route n'existe pas dans la table de routage, on en crée une nouvelle :

- On définit la passerelle (machine émettrice du message) et l'interface de la machine.
- On prend la nouvelle métrique
- On positionne le drapeau de changement de route
- On initialise le temporisateur de validité à 180 secondes

Finalement, si au moins une route a été modifiée ou a été ajoutée à la table de routage, on demande à la machine de déclencher une mise à jour.

Code : Vous pouvez consulter la méthode *receiptRIPDatagram(Datagram datagram, Interface interf)* de la classe *Computer*.

3.5 Modes du protocole RIP

Nous avons mis en place trois modes de fonctionnement du protocole RIP dans notre application :

- Soit il est actif, dans ce cas la machine pourra déclencher des mises à jour et tenir compte des messages RIP reçus pour modifier sa table de routage
- Soit il est actif en mode silencieux, la machine pourra simplement recevoir les messages RIP et les prendre en considération
- Dans le cas ou RIP est inactif, une machine ne pourra ni déclencher des mises à jour, ni être à l'écoute des messages de mise à jour RIP.

3.6 Extensions pour la version 2 du protocole

Comme on peut le lire dans le protocole de la version 2 : *"Le protocole RIP-1 actuel ne prend pas en compte les systèmes autonomes et les interactions IGP/EGP, le découpage en sous-réseaux, et l'authentification puisque ces implémentations postdatent RIP-1."* Nous avons donc ajouté les deux choses suivantes à notre application pour implémenter la version 2 :

- La possibilité d'associer n'importe quel masque à n'importe quel réseau pour pouvoir faire du découpage en sous-réseaux. Pour cela, nous avons ajouté à chaque entrée d'un datagramme RIP le masque correspondant au réseau (Fig. 15)
- L'identification d'un datagramme RIP en ajoutant une entrée d'identification (Fig. 16), même si cela n'est pas vraiment utilisé dans notre application puisque chaque machine utilise le même mot de passe

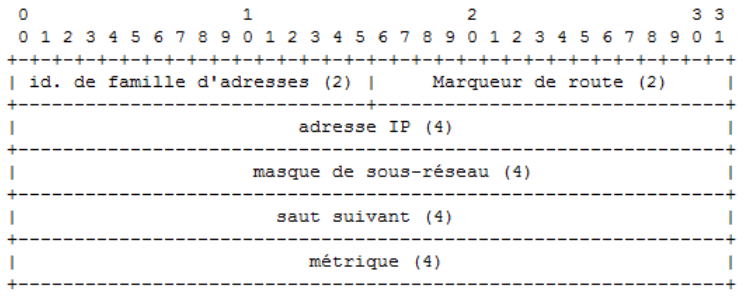

Fig. 15 – Format d'un message RIP version 2

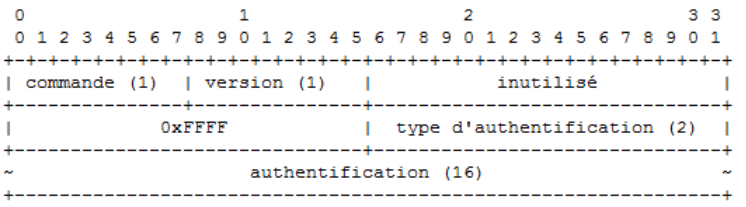

Fig. 16 – Entrée d'identification du datagramme RIP