Enoncé du TP 5 Réseaux

Fragmentation/Réassemblage de datagrammes IP, Analyse/production de trames

C. Pain-Barre

INFO - IUT Aix-en-Provence

version du 6/4/2012

---

1 Analyse et production de trames

Cette partie est consacrée à l'analyse et à la production de trames Ethernet V2. Ces opérations se feront autant manuellement qu'en utilisant des outils destinés à cela. La démarche manuelle devrait permettre d'appréhender concrètement le multiplexage/démultiplexage ainsi que l'encapsulation/décapsulation opérée par les couches. Ceci fait, nous pourrons utiliser des outils qui automatisent cette (fastidieuse) tâche. Ces outils ne seront utilisables que sur une machine virtuelle Linux que nous allons créer pour l'occasion, et sur laquelle vous serez administrateur (`root`).

Exercice 1 (Démarrage d'une machine virtuelle Debian)

Suivre les étapes suivantes afin de démarrer la machine virtuelle :

1. Dans un terminal, taper la commande suivante :

   ```
   $ wget http://infodoc/~cpb/enseignement/reseaux/tp/tp5/vm/startdebian.sh
   ```

   afin de télécharger le fichier `startdebian.sh`.

   > ⓘ C'est un script Bash qui automatise la création et le démarrage d'une machine virtuelle Debian.

2. Taper ensuite la commande :

   ```
   $ bash startdebian.sh
   ```

   qui exécute le script.

   > ✍ Son exécution prend du temps car il faut reconstruire le disque de la machine virtuelle pour la créer. Un indicateur de progression informe sur le temps restant pour cette opération, affiché à droite (`ETA ...`).
   > En attendant, **sauter la fin de l'exercice et commencer la partie «*Analyse de trames*».** Reprendre l'exercice lorsqu'elle aura démarré.

3. La machine virtuelle démarre dans une nouvelle fenêtre (ne pas modifier sa taille pour le moment). Cliquer *Ok* au message d'information qui est éventuellement affiché et laisser la machine démarrer sur le premier Linux proposé (ou taper *Entrée* pour aller plus vite).

   > ⓘ Le démarrage de Linux prend aussi un peu de temps...

4. Cliquer *Ok* aux messages d'information éventuellement affichés. Logez-vous en tant que `root` avec le mot de passe **`<re>zo++`** et cliquer sur *Suivant* au message d'alerte qui sera affiché.
5. La machine virtuelle est prête. Nous l'appellerons par la suite VM.

[Corrigé]

1.1 Analyse de trames

Dans cette partie, nous allons analyser le contenu de trames Ethernet V2 qui ont été "capturées". Chacune a été stockée dans un fichier texte contenant ses octets, écrits sous forme de nombres hexadécimaux (2 *digits* hexadécimaux par octet).

> ⓘ Ces trames ont été "capturées" par l'utilitaire **tcpdump** sous Linux, généralement réservé à `root`, qui existe aussi sous Windows. De nombreux utilitaires de captures existent, parmi lesquels Wireshark (Linux/Windows) et Windump (autre portage de **tcpdump** sur Windows)...

L'analyse d'une trame consiste à en extraire le plus d'information possible, notamment en procédant au démultiplexage et à la décapsulation que doit réaliser chaque couche concernée par la trame, à commencer par la couche Liaison d'Ethernet et, *in fine*, **retrouver quel est le protocole de plus haut niveau à l'origine de la trame, ainsi que l'objet du message qu'il a émis**.

En effet, les trames sont émises par la couche Liaison pour véhiculer des messages pour le compte de protocoles de niveau supérieur (de la couche 2 ou 3), tels que ARP, RARP, IP, XNS, IPX, ... Dans une trame Ethernet V2, le champ ***EtherType*** (ou *Type*) indique le protocole destinataire du message véhiculé. Lorsqu'une trame est reçue sans erreur, Ethernet remet les octets contenus dans son champ *Données* au (processus chargé du) protocole indiqué par l'*EtherType*. Ce protocole, situé au dessus d'Ethernet (dans la couche Liaison ou Réseau), traite ces octets comme un message conforme à ce protocole. Ainsi, durant l'analyse le champ *Données* de la trame doit être traité comme un message du protocole indiqué par son champ *EtherType*.

> ✍ Le format d'une trame Ethernet V2 est précisé dans le document ethernet.pdf, disponible dans la rubrique *Documents Utiles* du site `http://infodoc.iut.univ-aix.fr/~cpb/enseignement/reseaux`. Vous trouverez aussi dans cette rubrique le format des messages des différents protocoles que nous étudierons dans les exercices :
>
> - Address Resolution Protocol (ARP)
> - Internet Protocol (IP)
> - Internet Control (and Error) Message Protocol (ICMP)
>
> Bien entendu, toutes ces informations figurent aussi dans les transparents des cours qui leur sont consacrés.

Parmi les protocoles situés immédiatement au dessus d'Ethernet, certains (comme IP, IPX ou XNS) transportent aussi (encapsulent) des messages pour le compte d'autres protocoles comme ICMP, UDP ou TCP. De même, les protocoles TCP et UDP transportent (encapsulent) des messages pour le compte de protocoles d'**applications**, etc.

> ⓘ Nous étudierons les protocoles UDP et TCP, ainsi que certaines applications, un peu plus tard. Nous nous limiterons pour le moment aux protocoles Ethernet, ARP, IP et ICMP.

L'analyse consiste donc à remonter dans les couches, l'une après l'autre en décapsulant les messages véhiculés, jusqu'à parvenir à la couche de plus haut niveau (il n'y a plus de décapsulation à opérer) et l'interprétation du message qu'elle a émis. Durant l'analyse, chaque fois qu'un message d'une couche est traité, **il faut interpréter les informations contenues dans ses champs et les présenter sous une forme adaptée à leur nature.** Par exemple, une adresse Ethernet devra être présentée sous la forme commune des adresses Ethernet ; les adresses IP doivent être présentées en notation décimale pointée ; s'il y a démultiplexage il faut indiquer le nom du protocole concerné par les données ; s'il y a un *Checksum*, il faut indiquer s'il est correct, etc.

Ni le préambule ni le CRC des trames Ethernet ne figurent dans les captures :

- **le premier octet appartient à l'adresse Ethernet de destination ;**
- **le dernier appartient aux données de la trame.**

Les trames à analyser sont aussi disponibles dans le répertoire `~cpb/public/tpres/tp5/trames` sur allegro.

Exercice 2 (Analyse manuelle de la première trame capturée)

La première trame Ethernet V2 capturée est la suivante (fichier `trame_1.txt`) :

```
ffff ffff ffff 09ab 14d8 0548 0806 0001
0800 0604 0001 09ab 14d8 0548 7d05 300a
0000 0000 0000 7d12 6e03
```

La trame est présentée par groupes de 2 octets (avec 2 digits hexadécimaux par octet) séparés par des blancs ou des retours à la ligne (qui ne sont pas significatifs et servent juste à la lisibilité).

1. Analyser manuellement cette trame en commençant par identifier/extraire ses différents champs (Adresses, EtherType, Données) et les présenter sous une forme adaptée, dépendant de la nature du champ. Le champ *EtherType* doit permettre d'identifier le protocole destinataire des données de la trame.
   Analyser alors ces données comme un message de ce protocole, et identifier/extraire ses différents champs et les présenter sous une forme convenable. À nouveau, si ce protocole encapsule un message d'un protocole de niveau supérieur, procéder à sa décapsulation et son analyse, et ainsi de suite, jusqu'à ce qu'il n'y ait plus de décapsulation possible.
2. La question précédente a permis de déterminer le protocole de plus haut niveau à l'origine de cette trame. Quel est l'objet du message qu'il a transmis ? Notez-vous des incohérences dans la valeur d'un champ ?
3. Dans la VM (machine virtuelle), télécharger le fichier `trame_1.pcap` en ouvrant un navigateur, ou depuis un terminal en tapant la commande suivante :

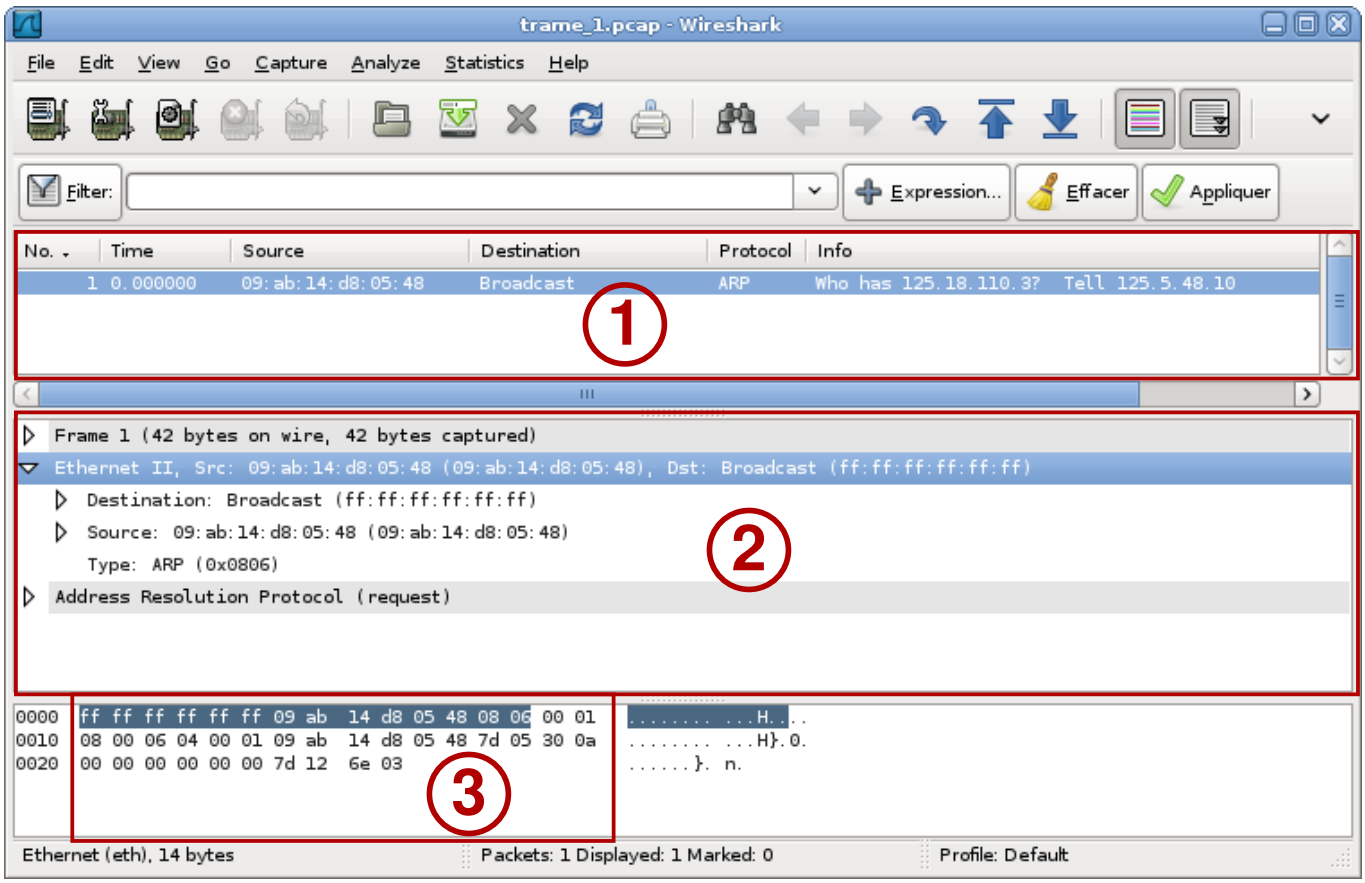
   ```
   # wget http://infodoc/~cpb/enseignement/reseaux/tp/tp5/trames/trame_1.pcap
   ```

   Ce fichier contient la trame codée dans le format PCAP, compréhensible par de nombreux outils d'analyse, dont **Wireshark**.

4. Toujours dans la VM, depuis le menu *Applications* $\longrightarrow$ *Internet*, lancer **Wireshark** (n'importe lequel), un outil célèbre d'analyse et de capture de trafic réseau.

5. Dans **Wireshark**, choisir le menu *File* ⟶ *Open* et sélectionner le fichier `trame_1.pcap` téléchargé précédemment. **Wireshark** analyse alors la trame et la présente dans une interface similaire à (agrandir éventuellement la fenêtre et les zones mises en évidence) :



- dans la zone ① sont listées les trames analysées, à raison d'une trame par ligne. Les colonnes donnent des informations essentielles telles que la source, la destination, le protocole et dans la colonne info, l'objet du message s'il peut être déterminé ;
- la zone ② détaille l'analyse de la trame sélectionnée dans la zone ①. Dans cette zone, en cliquant sur le marqueur ▷ qui débute une ligne, on développe les informations qu'elle regroupe (et le marqueur passe à ▽). Sur la figure :
  - ◇ la première ligne est l'état "brut" : une trame (*Frame*) avec le nombre d'octets qui la constituent ;
  - ◇ la deuxième ligne indique qu'il s'agit d'une trame Ethernet v2. Elle est développée et l'on voit le contenu de ses champs sur les lignes qui suivent ;
  - ◇ la dernière ligne montre que la trame contient un datagramme ARP. Cette ligne n'est pas (encore) développée.
- la zone ③ met en évidence les octets de la trame ou de l'information sélectionnée dans la zone ②. Ici, nous voyons que la ligne Ethernet II est sélectionnée dans la zone ②, et les octets de l'en-tête Ethernet V2 sont mis en évidence dans la zone ③.

> ⓘ À la droite de la zone ③, ces octets sont interprétés comme des caractères ASCII et sont affichés.

6. Vérifier que les informations que vous avez extraites sont conformes à ce qu'indique **Wireshark**. Vérifier aussi que la colonne *Info* de la zone ① correspond à ce que vous avez déduit.

[Corrigé]

Exercice 3 (Analyse automatisée de la deuxième trame capturée)

La deuxième trame capturée est la suivante (fichier `trame_2.txt`) :

```
000f 1f13 349a 0001 304a 3800 0800 4500
0054 9c1e 0000 3301 2d8c 8b7c bb04 ac10
cb6d 0000 f72b ea30 0002 c31f 6047 0e37
0200 0809 0a0b 0c0d 0e0f 1011 1213 1415
1617 1819 1a1b 1c1d 1e1f 2021 2223 2425
2627 2829 2a2b 2c2d 2e2f 3031 3233 3435
3637
```

Suivre les étapes suivantes pour procéder à une analyse partiellement manuelle, puis automatisée :

1. Sans extraire tous les champs de la trame, identifier le protocole destinataire de ses données et extraire le champ Données ;
2. Sans extraire tous les champs de ce protocole, identifier s'il encapsule un message d'un autre protocole. Si oui, extraire le champ Données et recommencer le processus. Le but est ici de repérer rapidement le protocole de plus haut niveau impliqué dans la trame, en se concentrant uniquement sur les champs nécessaires à cela, et laissant de côté les autres champs ;
3. Dans la VM, télécharger le fichier `trame_2.pcap` en ouvrant un navigateur, ou depuis un terminal en tapant la commande suivante :

   ```
   # wget http://infodoc/~cpb/enseignement/reseaux/tp/tp5/trames/trame_2.pcap
   ```

   puis l'ouvrir dans **Wireshark**.
4. Vérifier que vous aviez identifié les bons protocoles, puis déterminer l'objet du message émis par le protocole de plus haut niveau. Notez-vous des incohérences dans la valeur d'un champ ?
5. L'un des protocoles concernés est IP. Recalculer le *Checksum* IP à l'aide de l'utilitaire **checksum** disponible dans le répertoire `~cpb/public` sur allegro.

> ✍ On rappelle que le calcul du *Checksum* IP porte uniquement sur les octets de son en-tête (sans le champ *Checksum*). L'utilitaire **checksum** demande les octets (en hexadécimal) faisant l'objet du calcul et qui écrit le *checksum* calculé. Quelques infos sur son utilisation :
>
> - le retour à la ligne demande de procéder au calcul ;
> - les espaces sont ignorés ;
> - il faut que le nombre d'octets soit pair pour réaliser le calcul. Compléter si besoin par un dernier octet à 0.

[Corrigé]

Exercice 4 (Optionnel : Analyse manuelle de la deuxième trame capturée)

Cet exercice est facultatif

Effectuer entièrement manuellement l'analyse de la deuxième trame capturée.

[Corrigé]

1.2 Encapsulation de messages et production de trames

Dans ces exercices, il s'agit de produire la trame Ethernet V2 qui est émise pour véhiculer le message indiqué. Selon la nature de ce message, différents protocoles peuvent être utilisés pour l'encapsuler. La trame véhiculera alors tous les messages des protocoles impliqués dans ces encapsulations, de la même manière que les trames analysées précédemment.

> ✍ Il n'est pas demandé de faire figurer le *préambule* ni le *CRC* des trames.

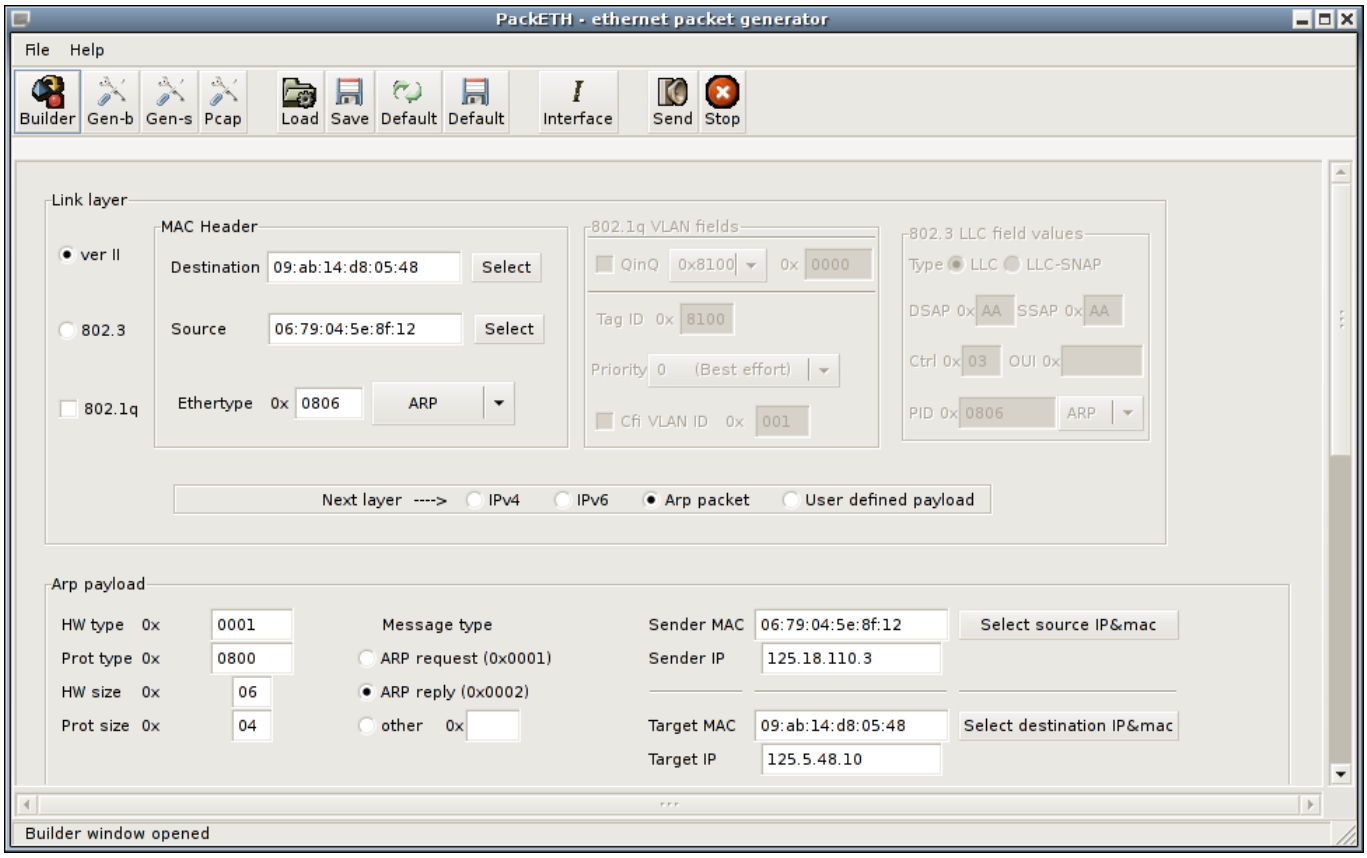
Exercice 5 (Première trame à produire manuellement)

Dans l'exercice 2 (la première trame analysée), la trame véhiculait une requête ARP demandant quelle était l'adresse Ethernet de la station d'adresse IP `125.18.110.3` :

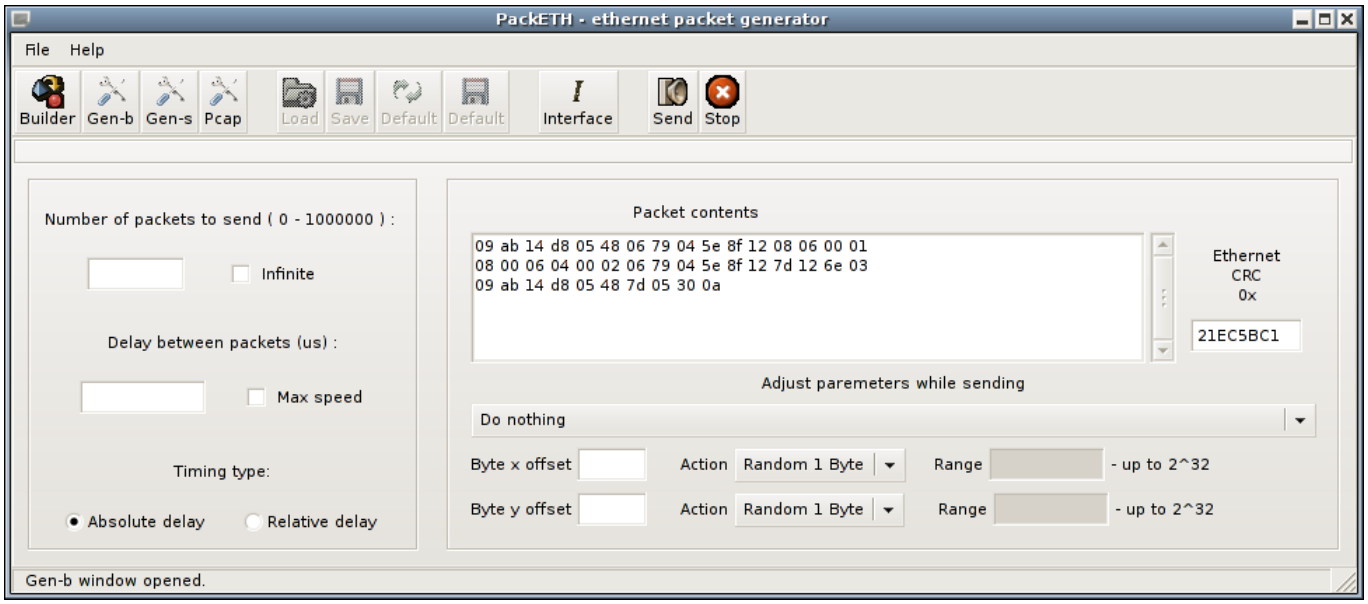
1. Produire la trame véhiculant la réponse ARP correspondante en supposant que cette adresse MAC est `06:79:04:5e:8f:12`. Pour cela, commencer par fabriquer le datagramme ARP, puis l'encapsuler dans la trame Ethernet V2 qui doit le véhiculer. Vous devez obtenir une suite d'octets similaire aux trames analysées précédemment.
2. Dans la VM, nous allons installer l'outil **packeth** qui automatise la création de trames. Ouvrir un terminal (dans la VM) et y taper les commandes suivantes :

```
# wget http://infodoc/~cpb/enseignement/reseaux/tp/tp5/vm/sources.list
# mv sources.list /etc/apt
# apt-get update
# apt-get install packeth
# packeth&
```

packeth doit s'exécuter et ouvrir une fenêtre de type :



3. Renseigner les différentes informations correspondant à la trame qui était à produire. Lorsque c'est fait, cliquer sur le bouton *Gen-b* en haut à gauche. Dans la zone centrale s'affichent les octets de la trame générée :



Comparer avec ce que vous avez produit.

[Corrigé]

Exercice 6 (Deuxième trame à produire automatiquement)

La trame analysée dans l'exercice 3 véhiculait un message ICMP de type ECHO REPLY émis par l'hôte `139.124.187.4` à destination de `172.16.203.109`. Ce type de message est principalement utilisé par la commande **ping** (vue plus tard). Il est généré en réponse à un message ICMP de type ECHO REQUEST.

Utiliser **packeth** pour produire la trame Ethernet V2 qui véhicule ce message ECHO REQUEST en supposant que le routeur que doit utiliser `172.16.203.109` pour joindre la station `139.124.187.4` a pour adresse physique `00:01:30:4a:38:00`. Les données de ce message sont les mêmes que celles de la réponse et les champs doivent correspondre à ceux de la réponse. Le datagramme IP contenant le message ICMP ne doit pas être fragmenté. Il a pour numéro d'identification 0 et une durée de vie de 64 (en décimal).

[Corrigé]

Exercice 7 (Optionnel : production manuelle de la deuxième trame)

Cet exercice est facultatif

Produire entièrement manuellement la trame demandée à l'exercice précédent.

[Corrigé]

2 Fragmentation et réassemblage de datagrammes IP

La taille du champ *Données* (charge utile ou *payload*) transporté par une trame est limitée. Cette limite s'appelle le ***Maximum Transfer Unit*** (MTU). Il peut être très différent selon les réseaux (1 500 octets pour Ethernet, 4 470 octets pour FDDI, ...). Le protocole IP, dans son rôle d'adaptation à tout type de réseau, tient compte de

leurs MTU. Lorsqu'il doit émettre un datagramme sur un réseau dont le MTU est inférieur à la taille de ce datagramme, il le fragmente pour émettre des datagrammes (fragments) n'excédant pas le MTU.

Cette adaptation est nécessaire pour les routeurs qui acheminent des datagrammes à travers divers réseaux, mais aussi pour les hôtes qui sont à l'origine d'un datagramme. En effet sur l'hôte source, quand un protocole (tel que ICMP, ou un protocole de transport comme UDP ou TCP [1]) demande à IP de transmettre un message, IP fabrique d'abord un datagramme qui encapsule le message dans son champ *Données*. Ensuite, selon le MTU du réseau que doit emprunter le datagramme (il n'y généralement qu'un seul réseau possible pour un hôte), IP le fragmente.

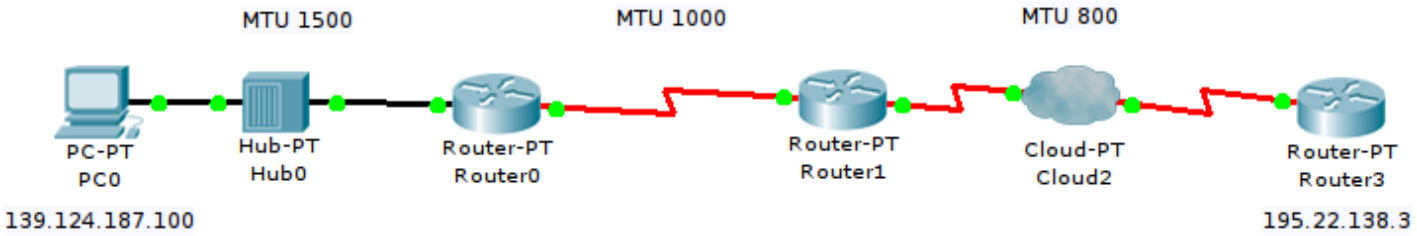
La technique de fragmentation est relativement simple : il s'agit de fabriquer autant de datagrammes IP qu'il est nécessaire pour transporter la totalité des données contenues dans le datagramme à fragmenter. Ces datagrammes fabriqués sont appelés les **fragments** du datagramme d'origine. Chaque fragment contient une partie des données du datagramme d'origine. Les fragments ont exactement le même format que les datagrammes IP (ce sont des datagrammes). Le bit *More* et le champ *Déplacement* indiquent s'il s'agit d'un fragment ou d'un datagramme complet.

Les fragments sont acheminés indépendamment les uns des autres à travers l'Internet. Il peuvent suivre des chemins différents et **peuvent être à leur tour fragmentés**.

> ✍ C'est la station destinataire du datagramme d'origine (et des fragments) qui doit réassembler les fragments pour reconstituer le datagramme d'origine avant d'en communiquer les données au protocole destinataire.

Exercice 8 (Démo de la fragmentation dans Packet Tracer)

Télécharger le fichier `tp5_lab1.pkt` puis l'ouvrir avec *Packet Tracer* (PT). Il contient 3 réseaux correspondant à la topologie suivante :



Comme indiqué sur le schéma, ces réseaux ont tous des MTU différents, modifiés pour les besoins de la démonstration : 1500, 1000 et 800, de la gauche vers la droite.

Ce lab a été préparé avec un scénario comprenant l'envoi d'un seul message ICMP (ECHO REQUEST) de source `139.124.187.100` (PC0) et destiné à `195.22.138.3` (Router3).
Ce message ICMP occupe au total 1480 octets. Le datagramme IP qui l'encapsule occupe 1500 octets.

Entrer dans le mode *Simulation*, et augmenter la vitesse de déroulement de la simulation. Cliquer ensuite sur *Auto Capture / Play* afin de commencer la simulation. Observer les différentes transmissions en remarquant que plusieurs datagrammes sont générés. Arrêter lorsque la réponse revient entièrement (marque verte) à PC0.

1. Cependant, le protocole TCP limite la taille de ses segments (MSS) pour éviter une fragmentation par l'hôte source.

> ⓘ Packet Tracer ne fragmente pas convenablement les datagrammes, et produit des tailles de datagrammes non conformes. C'est pourquoi nous ne nous intéresserons pas en détail au contenu des datagrammes transmis. Néanmoins, le nombre de fragments générés dans cet exemple est correct.

Explications sur le déroulement de cette simulation :

Le datagramme peut être émis sans fragmentation de PC0 vers Router0.
Une fois le datagramme reçu, Router0 doit le transmettre à Router1. Puisque le MTU de leur réseau est 1000, Router0 le fragmente et transmet 2 fragments :

(1) le premier contient 976 octets de données, pour une taille totale de 996 octets ;

(2) le second contient 504 octets de données, pour une taille totale de 524 octets.

Les deux datagrammes (fragments) reçus par Router1 doivent ensuite être transmis à Router3. Le fragment (2) peut être transmis sans problème, mais Router1 doit fragmenter le fragment (1) car sa taille excède le MTU de leur réseau. Ce datagramme (fragment) donnera lieu à 2 nouveaux fragments :

(1) le premier contient 776 octets de données, pour une taille totale de 796 octets ;

(2) le second contient 200 octets de données, pour une taille totale de 220 octets.

Au final, Router3 reçoit 3 fragments et reconstitue le message de 1480 octets qu'ils contiennent avant de le remettre à ICMP.

En réponse, Router3 envoie à PC0 un message ICMP (ECHO RESPONSE) de 1480 octets (1500 pour le datagramme) qui est fragmenté dès le départ en deux fragments :

(1) le premier contient 776 octets de données, pour une taille totale de 796 octets ;

(2) le second contient 704 octets de données, pour une taille totale de 724 octets.

Aucun de ces 2 fragments ne subira ensuite de fragmentation pour parvenir à PC0.

[Corrigé]

2.1 Technique de fragmentation

Lors de la fragmentation d'un datagramme, ses champs :

- *VER*
- *TOS*
- *Identification*
- bit *Don't Fragment*, qui doit être à 0 sinon on ne peut fragmenter
- *TTL*
- *Protocole*
- *Adresse IP Source*
- *Adresse IP Destination*

sont tous recopiés à l'identique dans l'en-tête des fragments générés.

Certaines options sont aussi recopiées dans tous les fragments (e.g. option *routage à la source*). D'autres ne le sont que dans le premier fragment (e.g. *enregistrement de route*). De ce fait, **la taille de l'en-tête des fragments (et donc le *champ HLEN*) peut varier d'un fragment à l'autre**.

✍ En l'absence d'option dans le datagramme d'origine, tous les fragments possèdent 20 octets d'en-tête.

Les autres champs (*Données*, *bit More*, *Déplacement*, *Longueur Totale*, *Checksum*) diffèrent entre les fragments.

La taille des données contenues dans chaque fragment ne doit pas excéder le MTU moins la taille de l'en-tête du fragment. En outre, elle doit être multiple de 8 (à cause du champ *Déplacement*), sauf pour le dernier fragment. IP choisit le plus grand multiple de 8, inférieur au MTU moins l'en-tête. Ainsi le nombre de fragments créés dépend du MTU, de la taille des données du datagramme d'origine et de la taille des en-têtes des fragments. Les fragments sont simplement générés les uns après les autres : le premier contenant le plus grand "morceau" possible des données à partir du début, le second contenant le plus grand "morceau" qui suit, etc.

Le bit *More* et le champ *Déplacement* (*Offset*) des fragments sont calculés en fonction du datagramme (ou fragment) que l'on fragmente, ainsi que du fragment créé. La figure 1 illustre le calcul de certains champs des 3 fragments d'un datagramme (ou fragment), formalisé ci-après.

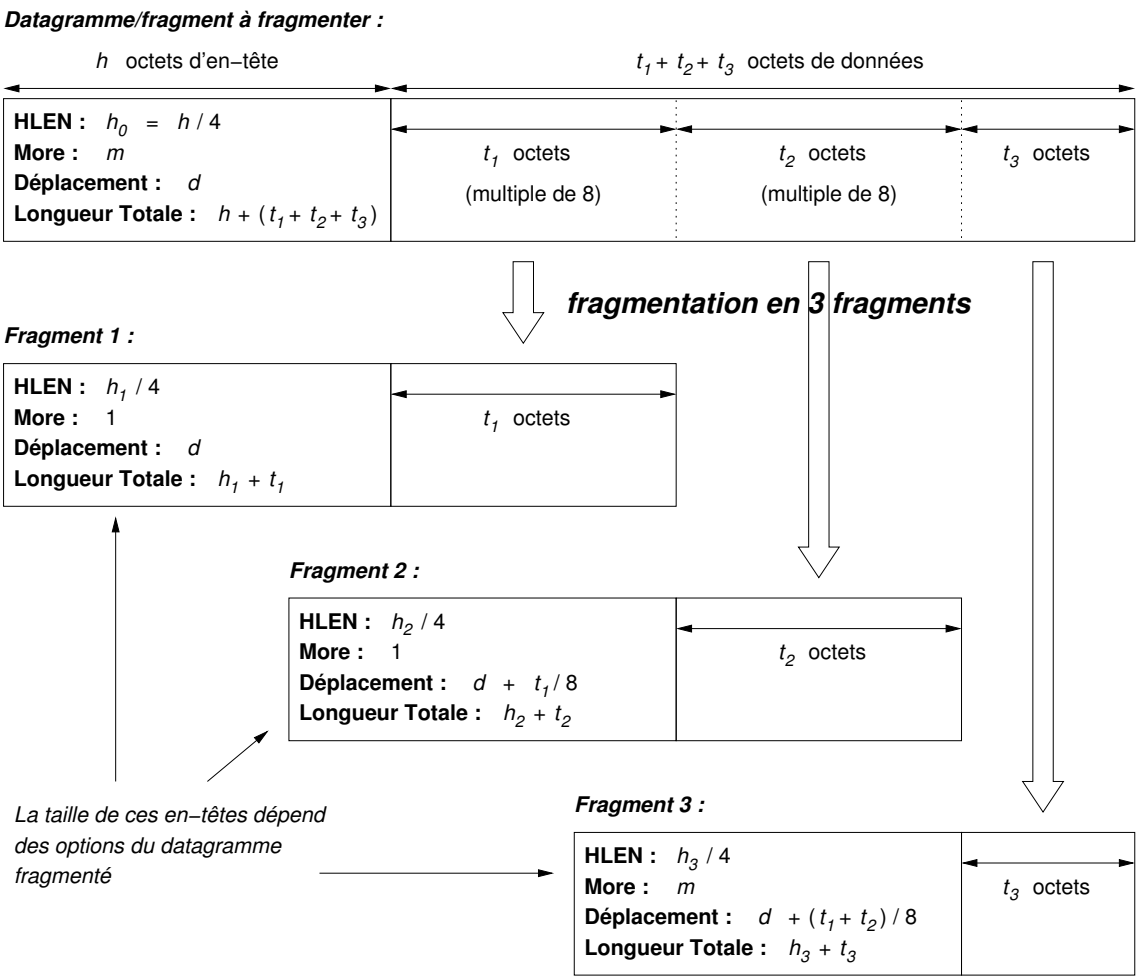


FIGURE 1 – Fragmentation d'un datagramme/fragment en 3 fragments

Prenons un datagramme (ou fragment) D qui doit être fragmenté en n fragments D_1, D_2,..., D_n. Posons m, la valeur du bit *More* de D et posons d, la valeur de son champ *Déplacement*. Pour un fragment D_i à générer, on

notera m_i la valeur de son bit *More*, d_i son champ déplacement et t_i la taille de ses données (et pas sa longueur totale). Les valeurs de m_i et d_i sont calculées comme suit :

- pour m_i : (indique si D_i est le dernier fragment ou s'il a une suite)
  - ◇ vaut 1 si $i < n$ (tous les fragments créés, sauf le dernier ont une suite) ;
  - ◇ vaut m si $i = n$ (le dernier fragment de D a une suite si D avait lui-même une suite) ;
- pour d_i : (position relative du premier octet de données de D_i dans les données du datagramme d'origine)

> ✍ Rappel : la position relative du premier octet de données de D est $d \times 8$. C'est pourquoi les fragments (sauf le dernier) doivent avoir une taille de données multiple de 8. Cela permet de limiter le nombre de bits occupés par le champ *Déplacement*.

  - ◇ vaut d si $i = 1$ (le premier octet de données de D_1 est le même et a la même position relative que le premier octet de données de D) ;
  - ◇ vaut $d + (\Sigma_{j=1}^{i-1} t_j)/8$ si $i > 1$ (la division par 8 vient du codage du champ *Déplacement*)

Les champs suivants sont aussi recalculés pour chaque fragment :

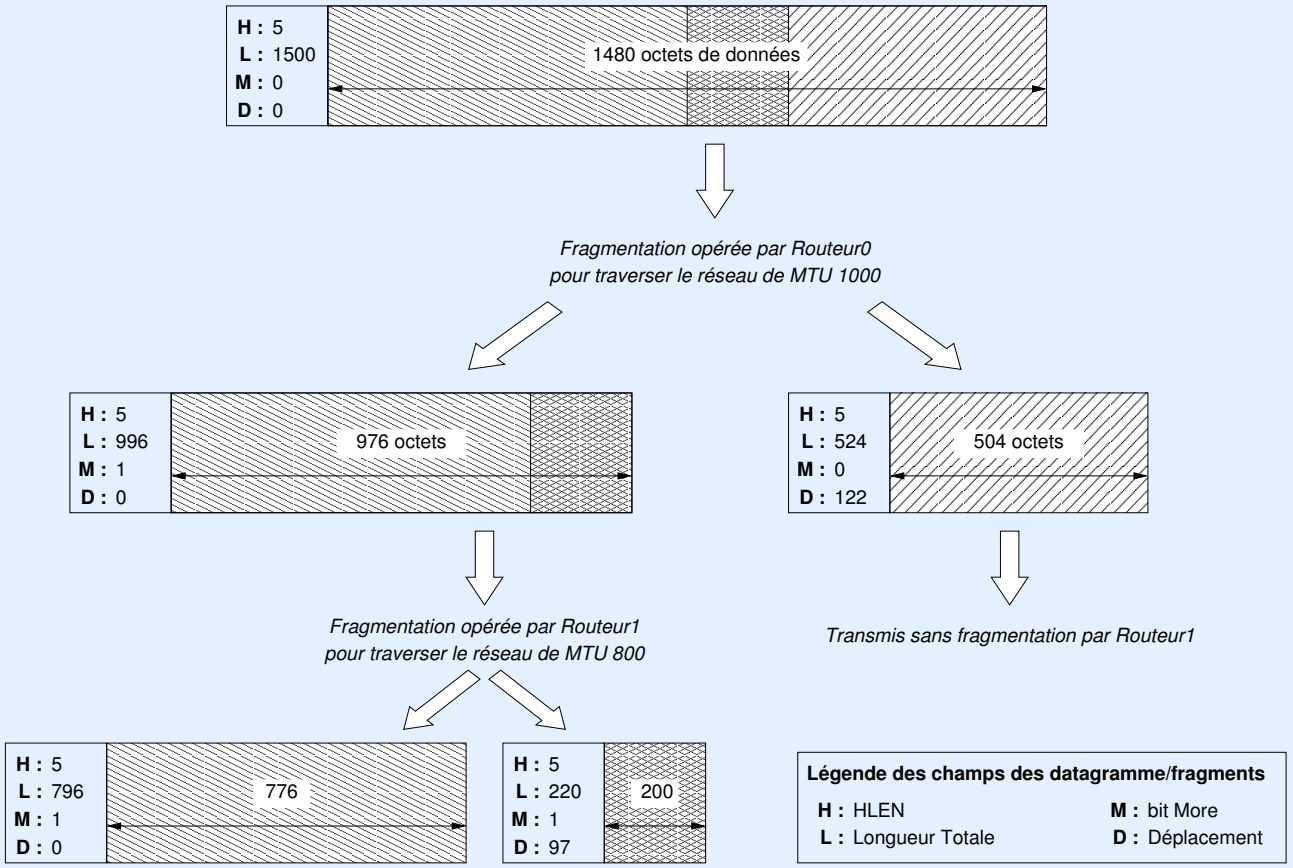
- *HLEN* : dépend des options ;

> ✍ Rappel : le champ HLEN est la taille de l'en-tête, divisée par 4 (pour gagner de la place sur son codage).

- *Longueur Totale* : taille en-tête + taille des données du fragment ;
- *Checksum* : recalculé à partir de l'en-tête du fragment.

Exemple 1

Dans l'exercice précédent, les fragmentations opérées étaient les suivantes :



Exercice 9 (Fragmentation de datagrammes)

Soit un hôte dont le logiciel IP doit envoyer un datagramme sans option contenant 5 000 octets de données à travers un réseau de MTU 1 800 :

1. Indiquer les champs suivants du datagramme d'origine :
   a. *HLEN*
   b. *Longueur Totale*
   c. bit *More*
   d. *Déplacement*
2. Fragmenter le datagramme et indiquer ces mêmes champs pour tous les fragments obtenus.
3. Le premier fragment et le troisième sont arrivés à un routeur qui doit les réexpédier par un réseau de MTU 1 000. Indiquer les champs précédents pour les fragments fabriqués par ce routeur.

[Corrigé]

2.2 Technique de réassemblage

Un hôte peut recevoir des fragments et des datagrammes complets provenant de plusieurs sources.

✍ On reconnaît un datagramme non fragmenté quand les deux conditions suivantes sont satisfaites :

- le bit *More* est à 0 ;
- le champ *Déplacement* vaut 0.

Dans ce cas, les données du datagramme peuvent être remises au protocole indiqué dans le champ *Protocole*.

Sinon, il s'agit d'un fragment et IP doit attendre que tous les fragments soient arrivés pour reconstituer le datagramme d'origine et remettre les données. Les fragments d'un même datagramme partagent les mêmes *Identification*, *Adresse IP Source*, *Adresse IP Destination* et *Protocole*. Ce quadruplet permet ainsi de reconnaître les fragments issus du même datagramme. IP regroupe les fragments qui possèdent le même quadruplet, et tente de reconstruire le bloc de données du datagramme d'origine en se basant sur le bit *More*, et les champs *HLEN*, *Longueur Totale* et *Déplacement* des fragments. La reconstitution aura abouti lorsque le fragment avec *Déplacement* à 0, et celui avec le bit *More* à 0 ont été reçus, et qu'il n'y a pas de "trous" dans le bloc de données reconstitué.

Puisque des fragments peuvent être perdus, le *TTL* des fragments reçus est décrémenté de 1 à chaque seconde passée sur l'hôte. Si l'un d'eux parvient à 0, tous les fragments possédant le même quadruplet sont détruits, et un message ICMP est envoyé à l'émetteur.

Exercice 10 (Fragmentation de datagrammes)

La station `139.134.187.4` a reçu 12 datagrammes dont les éléments importants de l'en-tête sont récapitulés dans le tableau ci-dessous (où *Num* est le numéro de réception du datagramme et *LT* est sa longueur totale) :

Num.	HLEN	LT	Proto	Ident	More	Dépl.	IP Source	IP Dest.
1	5	2420	17	12345	1	300	139.124.187.2	139.124.187.4
2	5	2420	17	54321	1	600	139.124.187.2	139.124.187.4
3	5	2420	6	12345	1	300	158.159.160.161	139.124.187.4
4	5	2420	6	12345	1	600	158.159.160.161	139.124.187.4
5	5	844	17	54321	0	1800	139.124.187.2	139.124.187.4
6	5	1380	17	12345	0	600	139.124.187.2	139.124.187.4
7	5	90	6	12345	0	1200	158.159.160.161	139.124.187.4
8	5	2420	6	12345	1	0	158.159.160.161	139.124.187.4
9	5	2420	6	12345	1	900	158.159.160.161	139.124.187.4
10	5	2420	17	12345	1	0	139.124.187.2	139.124.187.4
11	5	2420	17	54321	1	900	139.124.187.2	139.124.187.4
12	5	4820	17	54321	1	0	139.124.187.2	139.124.187.4

✍ Les valeurs dans les tableaux correspondent à ce qui est codé dans les champs. Notamment, la taille de l'en-tête est obtenue en multipliant *HLEN* par 4 ; la position relative du premier octet de données est obtenue en multipliant *Dépl.* par 8.

Analyser ces datagrammes pour reconstituer si possible le(s) bloc(s) de données du (des) datagramme(s) d'origine ou pour indiquer s'il manque des fragments.

Pour donner un sens un peu pratique à l'exercice, ces datagrammes sont disponibles sous forme de fichiers texte, dont le contenu est de type :

```
----------------------------- en-tête -----------------------------------------
HLEN.............. : 5 (20 octets d'en-tête sans option)
Longueur totale.. : 2420
Identification... : 54321
Bit More......... : 1
Déplacement...... : 600
Adresse IP Source : 139.124.187.2
Adresse IP Dest.. : 139.124.187.4
------------------------------ données ----------------------------------------
... données du datagramme ...
...
```

En affichant le champ *Données* des datagrammes qui auront pu être reconstitués en totalité, on devrait voir apparaître une image ASCII Art. Ce champ *Données* est obtenu en concaténant, dans le bon ordre, le champ *Données* de ses fragments.

> ⓘ Un petit script appelé **dataglue** est disponible dans le répertoire `~cpb/public` sur allegro. Il affiche la concatènation des champs *Données* des datagrammes passés en arguments (dans l'ordre). En passant en arguments tous les fragments ordonnés d'un même datagramme, on obtient ainsi le bloc de données du datagramme d'origine ;-)
> Les images ASCII-Art contenues dans les datagrammes sont parfois volumineuses. Pour les visualiser correctement, il faut agrandir le terminal et réduire la taille des caractères du terminal. Sur la Debian, les terminaux **gnome-terminal** permettent de zoomer en arrière par le menu *Affichage*.
> Enfin, certaines images sont bien mieux visualisables en *reverse vidéo* (caractères blancs sur fond noir) qu'on obtient facilement en sélectionnant les (lignes de) caractères sur le terminal.

Les datagrammes sont copiables depuis le répertoire `~cpb/public/tpres/tp5/datagrammes` sur allegro, en cliquant sur les liens ci-dessous, ou en récupérant l'archive datagrammes.tgz sur le site de cet enseignement :

- datagramme n° 1 (fichier `datagramme_01.txt`)
- datagramme n° 2 (fichier `datagramme_02.txt`)
- datagramme n° 3 (fichier `datagramme_03.txt`)
- datagramme n° 4 (fichier `datagramme_04.txt`)
- datagramme n° 5 (fichier `datagramme_05.txt`)
- datagramme n° 6 (fichier `datagramme_06.txt`)
- datagramme n° 7 (fichier `datagramme_07.txt`)
- datagramme n° 8 (fichier `datagramme_08.txt`)
- datagramme n° 9 (fichier `datagramme_09.txt`)
- datagramme n° 10 (fichier `datagramme_10.txt`)
- datagramme n° 11 (fichier `datagramme_11.txt`)
- datagramme n° 12 (fichier `datagramme_12.txt`)

[Corrigé]