

Enoncé du TP 4 Réseaux

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

C. Pain-Barre

INFO - IUT Aix-en-Provence

version du 7/3/2013

Table des matières

1	Analyse et production de trames	2
	Exercice 1 (Démarrage d'une machine virtuelle Debian)	2
1.1	Analyse de trames	2
	Exercice 2 (Téléchargement des fichiers de trames)	4
	Exercice 3 (Analyse manuelle de la première trame capturée)	4
	Exercice 4 (Vérification de l'analyse avec Wireshark)	5
	Exercice 5 (Analyse automatisée de la deuxième trame)	6
	Exercice 6 (Facultatif : Analyse manuelle de la deuxième trame)	7
1.2	Encapsulation de messages et production de trames	7
	Exercice 7 (Première trame à produire manuellement)	7
	Exercice 8 (Vérification de la trame produite avec packeth)	8
	Exercice 9 (Deuxième trame à produire automatiquement)	10
	Exercice 10 (Facultatif : production manuelle de la deuxième trame)	10
2	Fragmentation et réassemblage de datagrammes IP	11
	Exercice 11 (Démon de la fragmentation dans Packet Tracer)	11
2.1	Technique de fragmentation	13
	Exercice 12 (Fragmentation de datagrammes)	15
2.2	Technique de réassemblage	16
	Exercice 13 (Réassemblage des fragments de datagrammes)	16

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'automatiquement, avec les outils appropriés que sont **Wireshark** et **packeth**. La démarche manuelle devrait permettre d'appréhender concrètement le multiplexage/démultiplexage ainsi que les encapsulations/décapsulations opérées par les couches. Ceci fait, nous pourrons utiliser des outils qui automatisent cette (fastidieuse) tâche.


Exercice 1 (Démarrage d'une machine virtuelle Debian)

Wireshark n'étant pas disponible sur les postes de travail, nous allons créer et exécuter une machine virtuelle Linux où il est installé, et sur laquelle vous serez administrateur (root). Suivre les étapes suivantes afin de démarrer la machine virtuelle :


1. Dans un terminal, taper la commande suivante :

```
$ mkdebnat .bash
```

pour exécuter ce script Bash qui automatise la création et le démarrage d'une machine virtuelle Debian dotée de **Wireshark**.

 L'exécution du script 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 . . .). Une fois le disque reconstruit, la machine virtuelle démarrera et il faudra confirmer certains messages.
En attendant, **commencer la section 1.1** (*Analyse de trames*). Reprendre l'exercice pour le terminer lorsque la machine virtuelle aura démarré.

2. 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...

3. 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.
4. 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)...

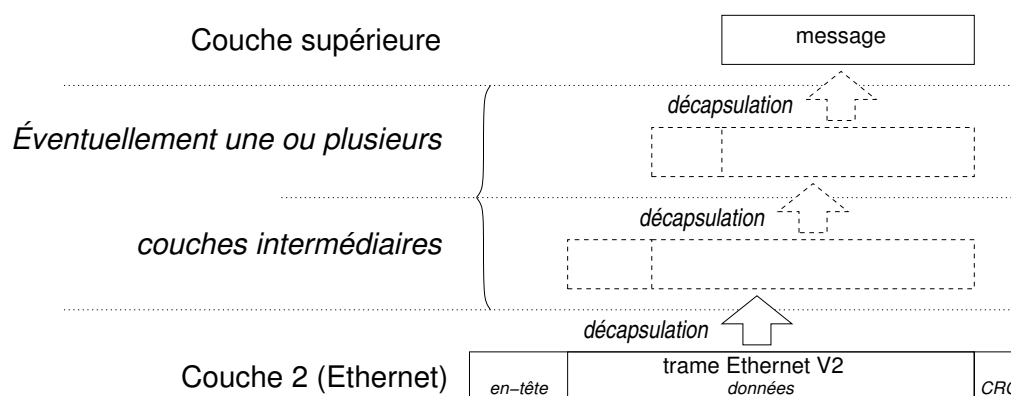


FIGURE 1 – Décapsulations (en nombre variable) à opérer pour extraire le message de niveau supérieur que la trame transporte.

L'analyse d'une trame consiste à en extraire le plus d'informations possible, notamment en procédant au démultiplexage et à la décapsulation que doit réaliser chaque couche concernée par (le contenu de) la trame, à commencer par la couche Liaison (dans notre cas, 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**. Autrement dit, l'analyse consiste à remonter dans les couches, l'une après l'autre en décapsulant les données des PDU (messages) des protocoles concernés, 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. Ce processus est schématisé dans la figure 1, où l'analyse commence en bas, par la couche 2, et se termine lorsqu'il n'y a plus de décapsulations à faire et que le message transporté peut être interprété.

En effet, une trame est émise par la couche Liaison (Ethernet) pour véhiculer un PDU (message) d'un protocole d'une couche supérieure, par exemple une application. En fonction de ce protocole, et de sa position (couche) dans l'architecture réseau, ce PDU n'est pas transporté (encapsulé) tel quel ¹ dans les données de la trame, mais peut être encapsulé dans le(s) PDU(s) d'une (plusieurs) couche(s) intermédiaire(s).

La trame (PDU d'Ethernet) encapsule dans son champ *Données* le PDU d'un protocole de la couche 3 ² (Réseau), tel que [ARP](#), [RARP](#), [IP](#), [XNS](#), [IPX](#), ... Dans une trame Ethernet V2, le [champ EtherType](#) (ou *Type*) indique le protocole destinataire des données de la trame. 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 de la couche 3 (Réseau) traite alors ces octets comme un PDU (message) de 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*.


Un PDU de la couche 3 (Réseau) peut éventuellement lui-même encapsuler, dans son champ *Données*, le PDU d'un autre protocole (de la même couche ou immédiatement supérieure). C'est le cas d'[IP](#), de [IPX](#) et de [XNS](#). En particulier, [IP](#) encapsule les PDU d'[ICMP](#) (couche 3), [IGMP](#) (couche 3), [UDP](#) (couche 4) et [TCP](#) (couche 4). De même, les protocoles de transport (couche 4) [TCP](#) et [UDP](#) encapsulent les PDU de protocoles d'**applications**, etc.

❗ Nous étudierons les protocoles de transport [UDP](#) et [TCP](#), ainsi que certaines applications, un peu plus tard. Nous nous limiterons pour le moment aux protocoles Ethernet, [ARP](#), [IP](#) et [ICMP](#), c'est à dire des messages n'excédant pas la couche 3 (Réseau).

1. Il se peut même que seule une partie du PDU d'origine se trouve dans la trame, comme nous le verrons dans la section 2.

2. Pour simplifier, par ce peut être un protocole de la couche 2 "supérieure".

Durant l'analyse, chaque fois que le PDU d'un protocole 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.

 Le format des PDU des protocoles étudiés est disponible dans les transparents des cours correspondants, ainsi que dans la rubrique *Documents Utiles* du site <http://infodoc.iut.univ-aix.fr/~cpb/enseignement/reseaux>, à raison d'un document par protocole :

- [Ethernet \(V2 et 802.3\)](#)
- [Address Resolution Protocol \(ARP\)](#)
- [Internet Protocol \(IP\)](#)
- [Internet Control \(and Error\) Message Protocol \(ICMP\)](#)

Exercice 2 (Téléchargement des fichiers de trames)

Sur le poste de travail, ouvrir un terminal et :

1. Récupérer l'archive `trames.tgz`, contenant les fichiers des trames de ce TP en tapant :

```
# wget http://infodoc/~cpb/enseignement/reseaux/2012/tp/tp4/enonce/trames.tgz
```

où, selon le cas, il faudra remplacer `infodoc` par `infodoc.iut.univ-aix.fr`. Alternativement, cette archive se trouve aussi dans `/commun/pain-barre/reseaux` ;

2. Extraire l'archive, ce qui crée le répertoire `trames` contenant 4 fichiers, en tapant :


```
# tar xzvf trames.tgz
```

[Corrigé]

Exercice 3 (Analyse manuelle de la première trame capturée)

La première trame Ethernet V2 capturée est contenue dans le 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). Ni le préambule ni le CRC des trames Ethernet ne figurent dans ces captures :

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

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 à 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. À part le préambule et le CRC, manque-t-il quelque chose d'autre à cette trame ?

❗ Il peut être utile de rappeler que la taille minimale d'une trame Ethernet, sans compter le préambule, est 64 octets...

[\[Corrigé\]](#)

Exercice 4 (Vérification de l'analyse avec Wireshark)

1. Dans la VM (machine virtuelle), recommencer l'exercice 2 pour télécharger et extraire l'archive des trames.
2. Toujours dans la VM, depuis le menu *Applications* → *Internet*, lancer **Wireshark** (n'importe lequel), un outil célèbre d'analyse et de capture de trafic réseau.
3. Dans **Wireshark**, choisir le menu *File* → *Open* et sélectionner le fichier `trame_1.pcap` extrait précédemment.

❗ Ce fichier contient la même trame qu'à l'exercice précédent, mais codée dans le format **PCAP**, compréhensible par de nombreux outils d'analyse, dont **Wireshark**.

Wireshark analyse alors la trame et la présente comme sur la figure 2 :

- 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.

4. 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.

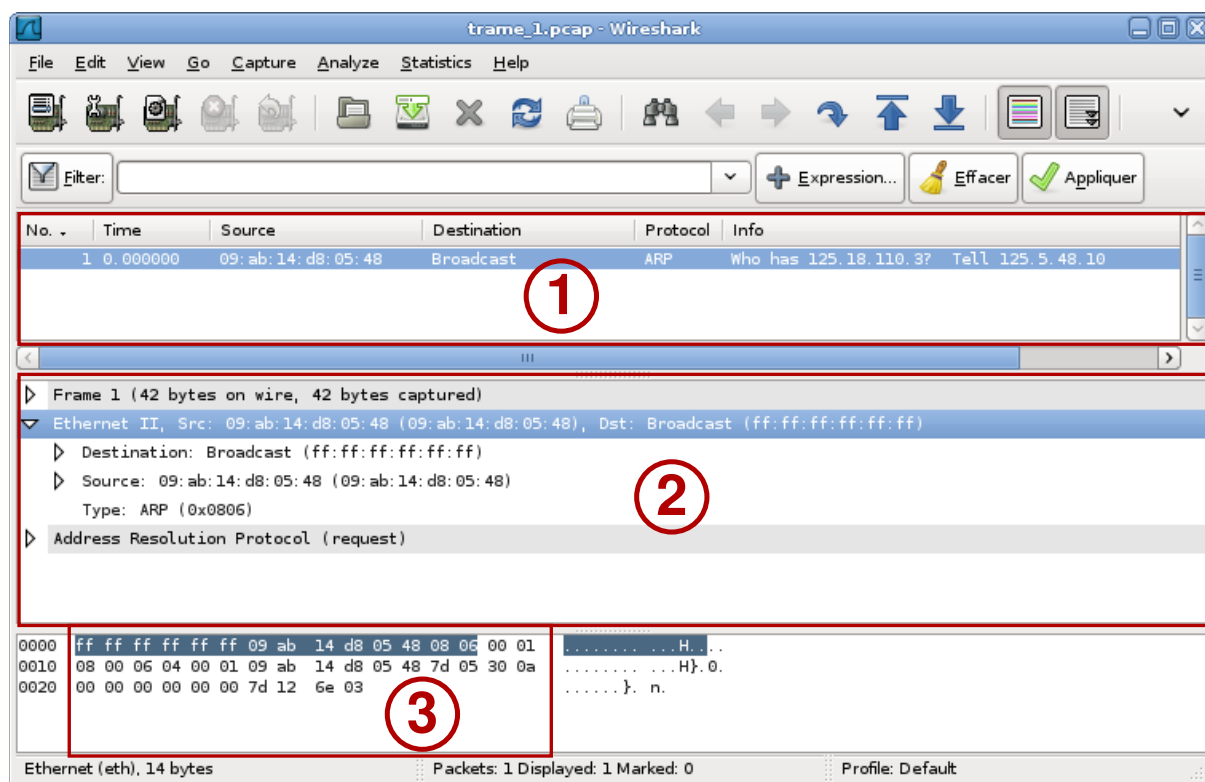


FIGURE 2 – Interface de Wireshark présentant clairement le contenu de la trame de l'exercice précédent.

[\[Corrigé\]](#)

Exercice 5 (Analyse automatisée de la deuxième trame)


La deuxième trame capturée est contenue dans le 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 manuellement à une analyse succincte, puis entièrement automatisée :

1. Sans extraire tous les champs de la trame, identifier le protocole destinataire des données et extraire le champ *Données* ;
2. Sans extraire tous les champs du message de ce protocole de la couche 3, 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 au démultiplexage, et laissant de côté les autres champs ;
3. Dans la VM, ouvrir le fichier `trame_2.pcap` 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. Observer les adresses IP et les adresses MAC. Que pouvons-nous raisonnablement en conclure sur l'adresse MAC source de la trame ?

6. L'un des protocoles concernés est IP. Recalculer le *Checksum* IP à l'aide de l'utilitaire **checksum** disponible dans le répertoire `/commun/pain-barre/reseaux`.

 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. Voici quelques précisions sur l'utilisation de **checksum** :

- saisir, sur une seule ligne, uniquement les octets faisant l'objet du calcul du checksum ;
- les espaces sont ignorés ;
- le retour à la ligne demande de procéder au calcul ;
- il faut que le nombre d'octets soit pair pour réaliser le calcul. Compléter si besoin par un dernier octet valant 0 (00 en hexa).

[\[Corrigé\]](#)

Exercice 6 (Facultatif : Analyse manuelle de la deuxième trame)

Cet exercice est facultatif

Effectuer entièrement manuellement l'analyse de cette deuxième trame.

[\[Corrigé\]](#)

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 un message. 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 7 (Première trame à produire manuellement)

Dans l'exercice 3 (la première trame analysée), la trame était émise en diffusion et véhiculait une **requête ARP** envoyée par l'hôte d'IP 125.5.48.10 et d'adresse MAC (Ethernet) 09:ab:14:d8:05:48, demandant l'adresse MAC de la machine ayant l'IP 125.18.110.3.

Produire la trame véhiculant la **réponse ARP** que doit renvoyer l'hôte 125.18.110.3 en supposant que son 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 produire une suite d'octets en hexa, similaire aux trames analysées précédemment.

[\[Corrigé\]](#)

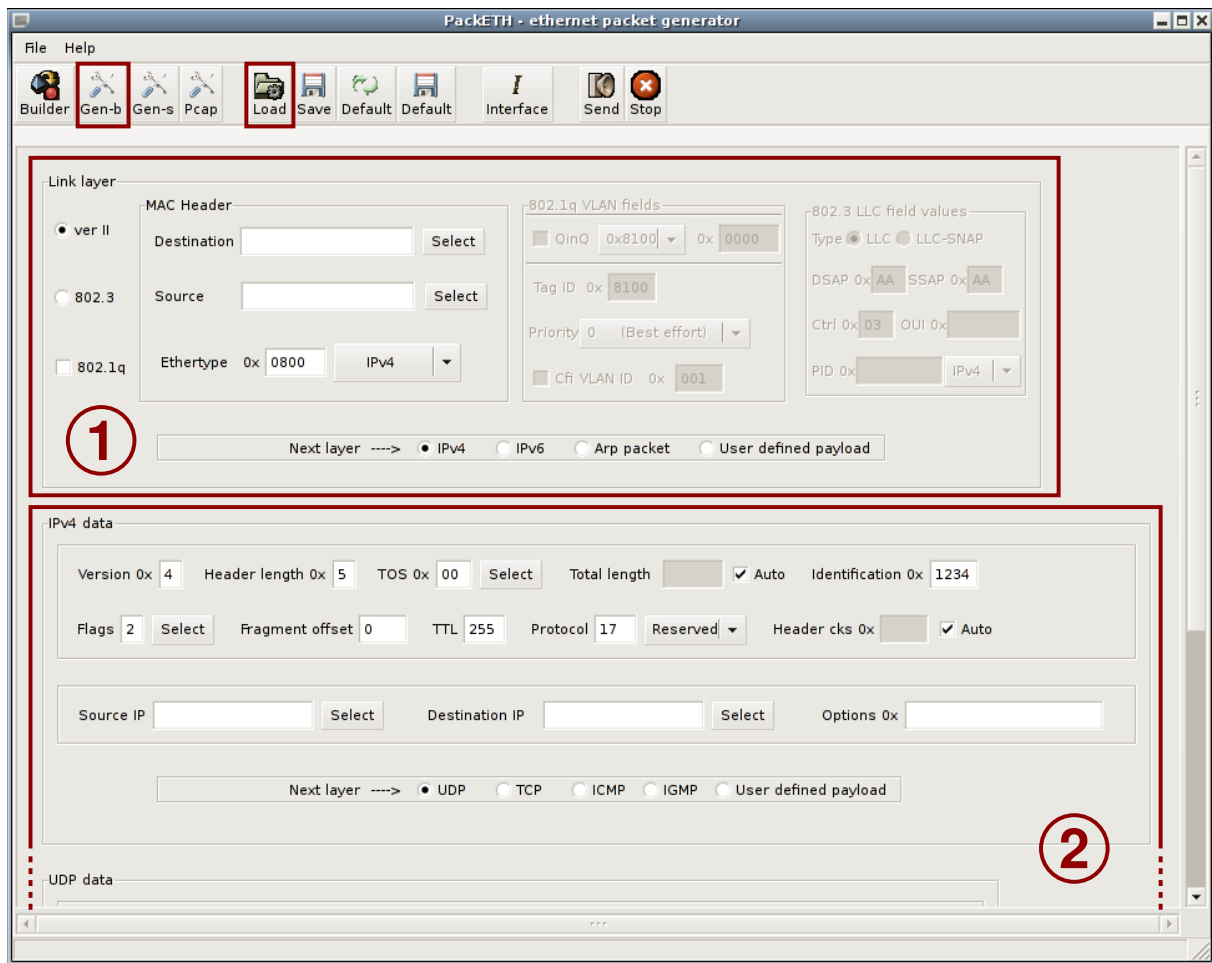
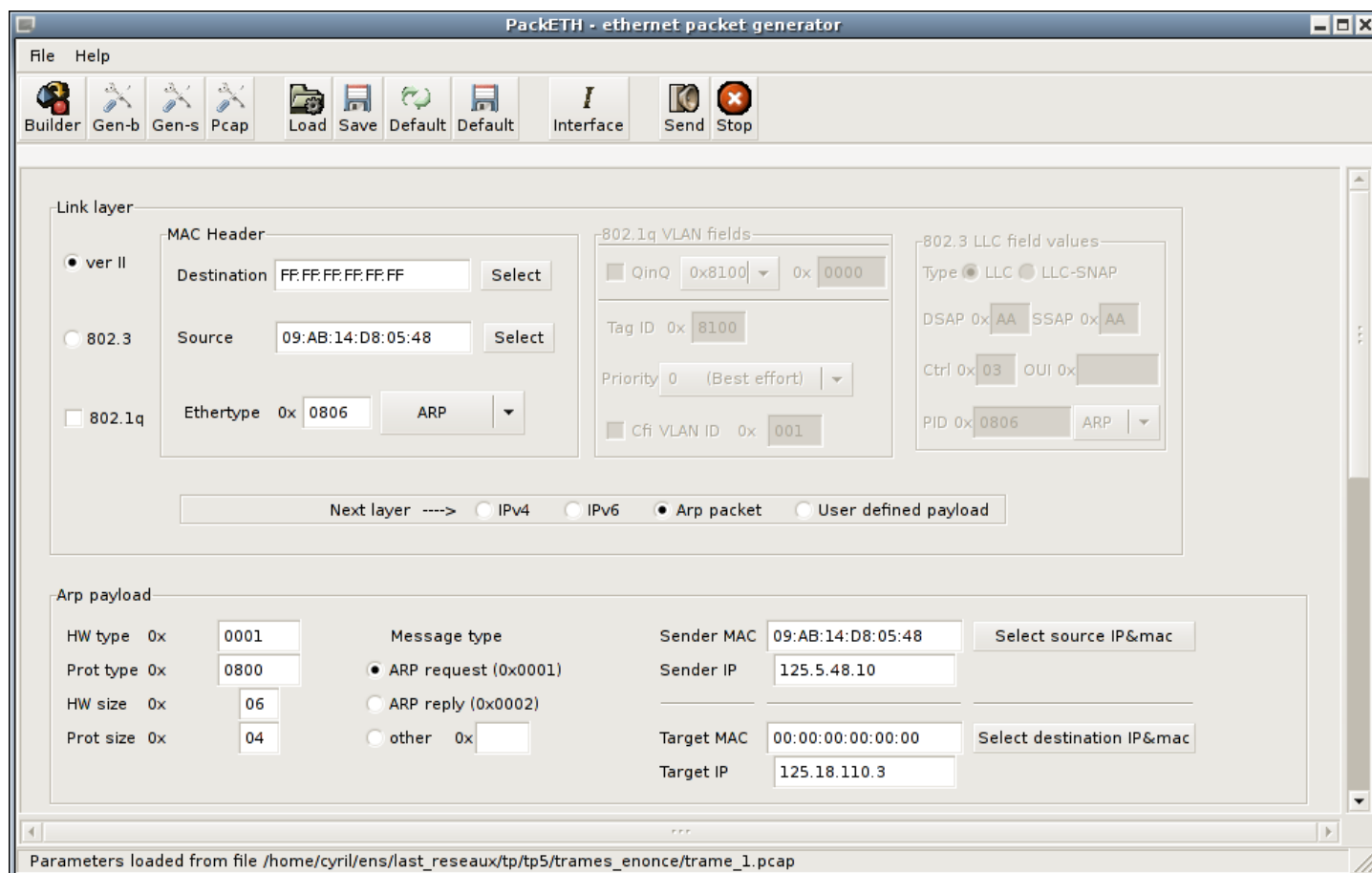


FIGURE 3 – Interface de **packeth** au démarrage, qui présente les zones de saisies pour Ethernet V2, IPv4 et UDP.

Exercice 8 (Vérification de la trame produite avec packeth)

L'outil **packeth** peut analyser/produire/envoyer des trames Ethernet. C'est une application graphique qui présente au démarrage l'interface de la figure 3 :

- en haut, la barre d'outils présente 2 boutons (mis en évidence) dont nous aurons besoin :
 - ◊ le bouton **Gen-b** servira à générer une trame sous la même forme que celles que nous avons analysées, à partir des informations saisies dans la zone centrale ;
 - ◊ le bouton **Load** permet de charger une trame au format PCAP pour détailler son contenu dans la zone centrale ;
- dans la zone centrale, la zone ① ou «*Link Layer*» présente l'en-tête de la trame (couche 2/Liaison). Sur la gauche, on peut choisir parmi l'en-tête Ethernet V2, IEEE 802.3 ou IEEE 802.1q, ce qui (dés)active les champs correspondants. Cette zone se termine par un choix *Next Layer* à faire parmi IPv4, IPv6, ARP ou autre. Ce choix détermine ce qui est affiché dans la zone ② ;
- la zone ② commence par une zone de saisie des champs de l'en-tête du protocole de la couche 3 (Réseau) qui a été choisi dans le *Next Layer* du niveau 2. Sur la figure, il s'agit d'IPv4, dont la zone se termine par un choix *Next Layer* à faire parmi UDP, TCP, ICMP, IGMP ou autre. Ici, le choix est UDP et la zone de saisie de l'en-tête UDP (couche 4/Transport) figure en dessous. Il n'y a pas de zone pour la couche 4 si le protocole de niveau 3 choisi est ARP ou autre.

FIGURE 4 – Présentation dans **packeth** de la requête ARP de l'exercice 3

i **packeth** est installé sur les postes de travail et dans la VM, et exécutable via la ligne de commande (**packeth**) ou via le menu *Applications* → *Internet* → *PackETH*.

1. Dans la VM, exécuter **packeth**. Constater que les zones affichées sont par défaut Ethernet V2, IPv4 et UDP;
2. Utiliser le bouton **Load** pour charger le fichier `trame_1.pcap` contenant la trame de la requête ARP de l'exercice 3 au format PCAP. L'interface s'adapte pour présenter cette requête ARP comme sur la figure 4. Observer le contenu des différents champs;
3. Modifier ce qu'il faut pour que la trame corresponde à la réponse ARP de l'exercice précédent. 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. La figure 5 met en évidence cette zone, pour la requête ARP. Comparer les octets générés pour cette réponse ARP avec la trame que vous avez produite à l'exercice précédent.

[Corrigé]

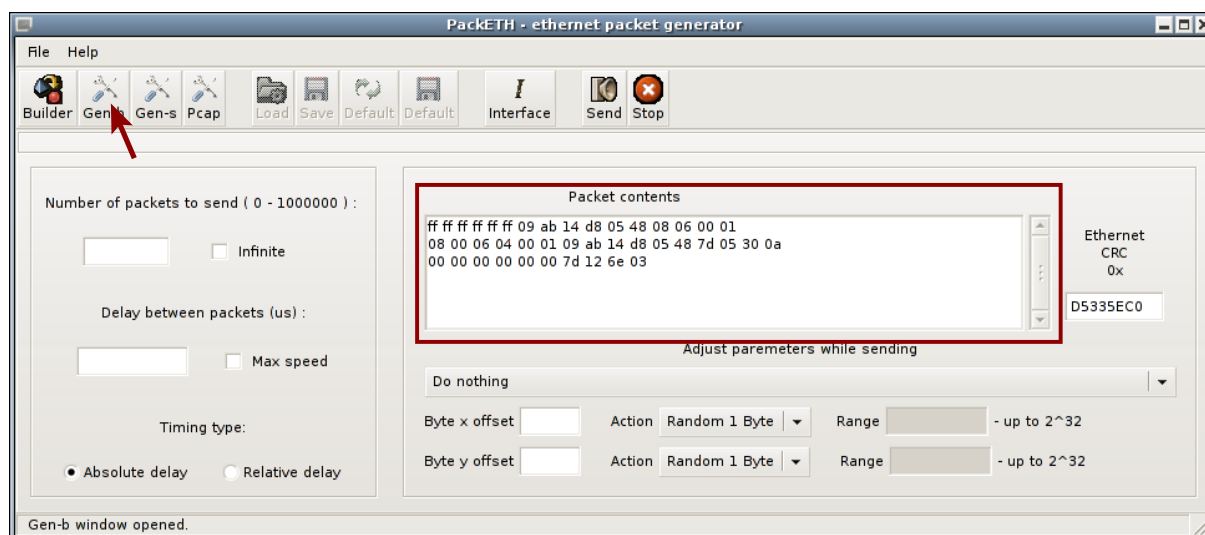


FIGURE 5 – Visualisation dans **packeth** des octets qui constituent la requête ARP de l'exercice précédent (avec un petit décalage dû à la suite de f...)

Exercice 9 (Deuxième trame à produire automatiquement)

La trame analysée dans l'exercice 5 véhiculait un message ICMP de type ECHO REPLY émis par l'hôte 139.124.187.4 à destination de 172.16.203.109. Il est généré en réponse à un message ICMP de type ECHO REQUEST. Cet échange de messages sert notamment à tester la connectivité entre deux hôtes, et c'est ce qui se produit lorsque nous le faisons avec *Packet Tracer*. Sur un hôte, la commande **ping** (vue plus tard) est généralement disponible pour faire ce test, avec une succession d'échanges.

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* du message ICMP, ainsi que l'*Identifiant* et le *Numéro de Séquence*, sont les mêmes que dans 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).

Pour vous faciliter la tâche, vous pouvez charger le fichier [trame_2.pcap](#) qui contient la trame de l'ECHO REPLY au format PCAP.

[Corrigé]

Exercice 10 (Facultatif : 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** ou **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 leur MTU. C'est un élément de la **configuration de l'interface** qui raccorde un hôte/routeur à un réseau. Il peut être précisé si besoin mais on se contente généralement des valeurs par défaut par type d'interface dont dispose le système. Lorsque IP doit émettre un datagramme sur un réseau dont le MTU est inférieur à la taille de ce datagramme, il le fragmente pour émettre à la place 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 TCP³) 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 a généralement qu'un seul réseau possible pour un hôte), IP le fragmente.

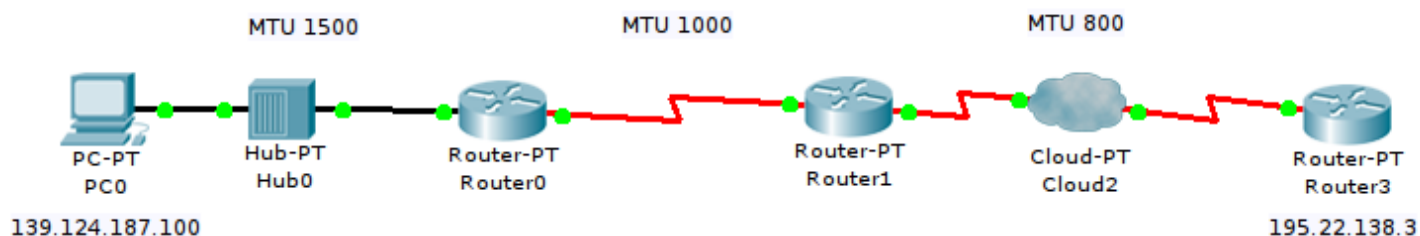
La technique de fragmentation est relativement simple : il s'agit de fabriquer autant de datagrammes IP qu'il est nécessaire pour transporter la totalité des données contenues dans le datagramme à fragmenter. Ces datagrammes fabriqués sont appelés les **fragments** du datagramme d'origine. Chaque fragment 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. Ils 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 11 (Démonstration de la fragmentation dans Packet Tracer)

Télécharger le fichier `tp4_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

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

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.

i 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 :

1. Le datagramme peut être émis sans fragmentation de PC0 vers Router0 ;
2. Une fois le datagramme reçu, Router0 doit le transmettre à Router1.
Le MTU de leur réseau étant 1000, Router0 le fragmente et transmet 2 fragments (avec les tailles correctes) :
 - Fragment 1 : contient 976 octets de données, pour une taille totale de 996 octets ;
 - Fragment 2 : contient 504 octets de données, pour une taille totale de 524 octets ;
3. 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 :
 - Fragment 1.1 : contient 776 octets de données, pour une taille totale de 796 octets ;
 - Fragment 1.2 : contient 200 octets de données, pour une taille totale de 220 octets ;
4. Au final, Router3 reçoit 3 fragments et reconstitue le message de 1480 octets qu'ils contiennent avant de le remettre à ICMP.
5. 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 :
 - le premier contient 776 octets de données, pour une taille totale de 796 octets ;
 - le second contient 704 octets de données, pour une taille totale de 724 octets.
6. Aucun de ces 2 fragments ne subira ensuite de fragmentation pour parvenir à PC0.

Les étapes des fragmentations opérées sont schématisées dans la figure 6, où l'en-tête comporte d'autres informations utiles étudiées dans la suite.

[\[Corrigé\]](#)

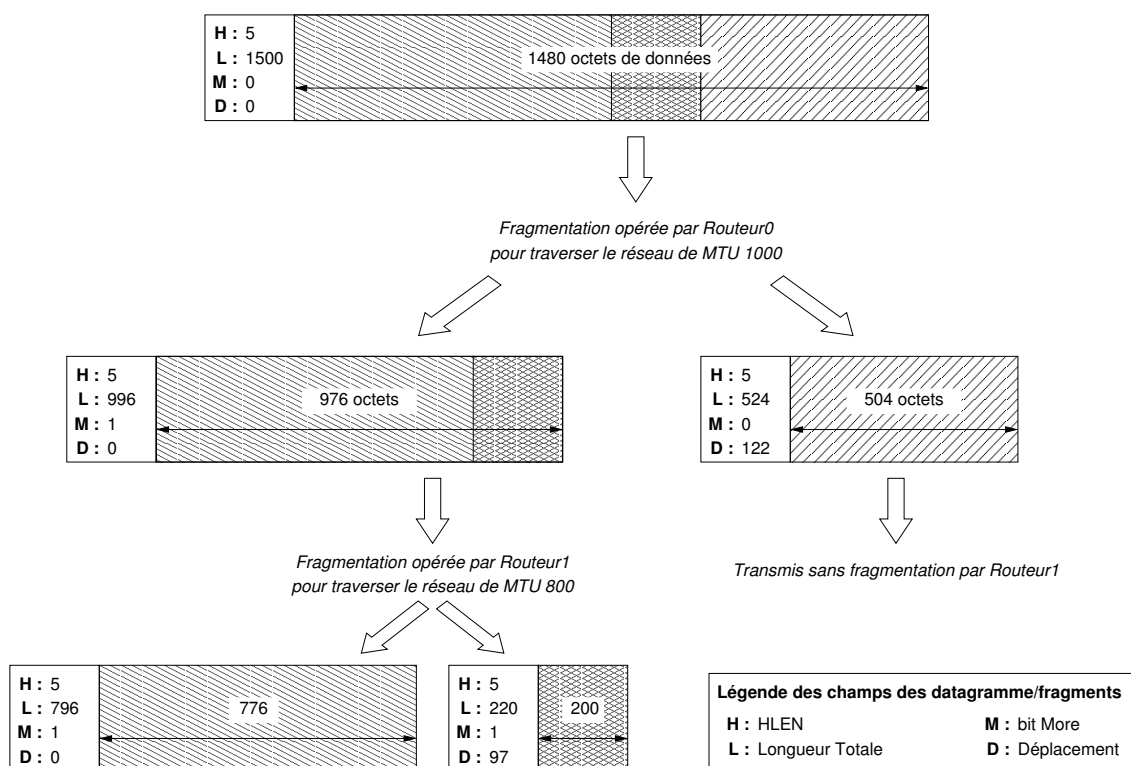


FIGURE 6 – Fragmentations du datagramme de l'exercice 11

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 il est interdit de fragmenter)
- *TTL*
- *Protocole*
- *Adresse IP Source*
- *Adresse IP Destination*

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

Certaines options sont aussi copiées dans tous les fragments (e.g. option *routing 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 à fragmenter et de la taille des en-têtes des fragments. Les fragments sont simplement générés les uns après les autres : le premier contient dans son champ *Données* le plus grand "morceau" possible (à partir du début) du champ *Données* du datagramme à fragmenter, le second contenant le plus grand "morceau" qui suit, etc.

Le bit *More* et le champ *Déplacement* (*Offset*) d'un fragment indiquent si ce fragment a une suite, et la position (numéro) qu'occupait le premier octet du champ *Données* dans le champ *Données* du datagramme non fragmenté. Ils sont facilement calculés durant la fragmentation, et sont utilisés pour le réassemblage.

Leur calcul est formellement détaillé dans ce paragraphe. Prenons un datagramme (ou fragment) D qui doit être fragmenté en n fragments D_1, D_2, \dots, 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 (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 $d + (\sum_{j=1}^{i-1} t_j)/8$ si $i > 1$ (la division par 8 vient du codage du champ *Déplacement*)

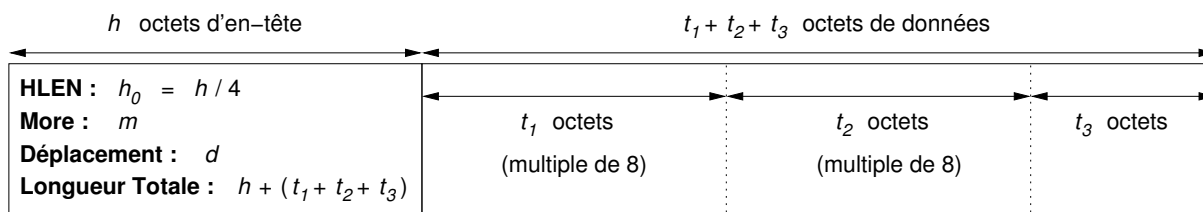
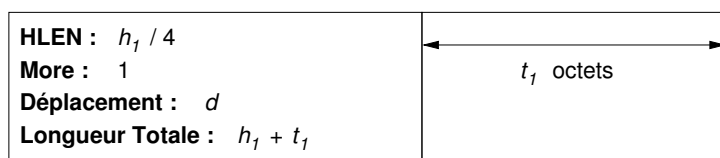
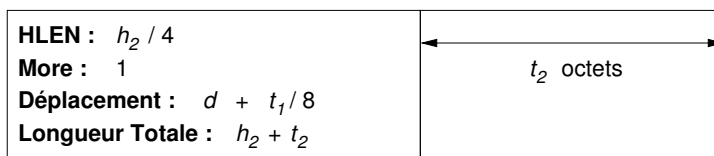
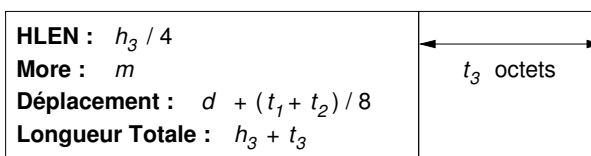
La figure 7 montre une illustration ce calcul, ainsi que de certains autres champs pour fragmenter un datagramme (ou fragment) en 3 fragments.

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.

Datagramme/fragment à fragmenter :**Fragment 1 :****Fragment 2 :****Fragment 3 :**

La taille de ces en-têtes dépend des options du datagramme fragmenté

FIGURE 7 – Fragmentation d'un datagramme/fragment en 3 fragments

Exercice 12 (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. Ce datagramme peut être fragmenté :

- Indiquer les champs suivants du datagramme d'origine :
 - HLEN*
 - Longueur Totale*
 - bit *More*
 - Déplacement*
- Fragmenter le datagramme et indiquer ces mêmes champs pour tous les fragments obtenus.
- 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 13 (Réassemblage des fragments 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* donne l'ordre de réception du datagramme ; les autres colonnes sont, dans l'ordre, les champs *HLEN*, *Longueur Totale*, *Protocole*, *Identification*, *bit More*, *Déplacement*, *Adresse IP Source* et *Adresse IP Destination*) :

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.

1. Analyser les champs de ces datagrammes et constater qu'il s'agit de fragments, générés par fragmentation d'un certain nombre de datagrammes d'origine (non fragmentés).
2. Regrouper les fragments par datagramme d'origine et les ordonner. Pour chaque datagramme d'origine, préciser s'il peut être reconstitué en totalité ou s'il manque (au moins) un fragment, puis en déduire la valeur des champs de son en-tête.
3. Pour donner un sens plus pratique et ludique à l'exercice, sa réalisation permet de découvrir quelques images ASCII-Art ;-) C'est pourquoi les 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 -----
... à partir d'ici, commencent les données ...
... du fragment/datagramme (une partie d'une image) ...
```

Télécharger l'archive [datagrammes.tgz](#) depuis le site et l'extraire, ce qui crée le répertoire `datagrammes` contenant les 12 fichiers datagrammes [datagramme_01.txt](#) à [datagramme_12.txt](#).

4. Les données d'un datagramme d'origine dont aucun fragment ne manque forment une image ASCII-Art. Afficher les images de ces datagrammes en concaténant, dans le bon ordre, le champ *Données* de leurs fragments. Pour automatiser cette tâche, utiliser le script **dataglué** présent dans `/commun/pain-barre/reseaux`. En lui passant en arguments tous les fragments ordonnés d'un même datagramme, il affiche le bloc de données du datagramme d'origine —dans notre cas, une image ASCII-Art. Par exemple, si `datagramme_01.txt`, `datagramme_02.txt` et `datagramme_03.txt` sont tous les fragments d'un même datagramme, générés dans cet ordre, alors :

```
$ dataglué datagramme_01.txt datagramme_02.txt datagramme_03.txt
```

affiche l'image.

 Ces images 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 toutes les lignes du terminal, avec la souris ou par le menu *Édition* → *Tout sélectionner*.

[Corrigé]