TP de Réseaux
N° 2

Etude de protocoles

INTRODUCTION

Concevoir un réseau, tirer des câbles, relier les machines, mettre à jour les fichiers de configuration,... Tout cela est nécessaire, mais ne suffit pas pour que deux machines communiquent.

D'autres problèmes sont à résoudre, notamment:

- Quelle technique utiliser pour avoir l'accès exclusif au médium;
- Quel chemin doit prendre le message pour parvenir à destination;
- Comment s'adresser à une application lancée sur une autre station;

etc…

L'objectif de ce TP est d'observer et de comprendre à l'aide de quels protocoles deux machines communiquent à travers un réseau.

Avant d'entamer les manipulations, voici quelques rappels :

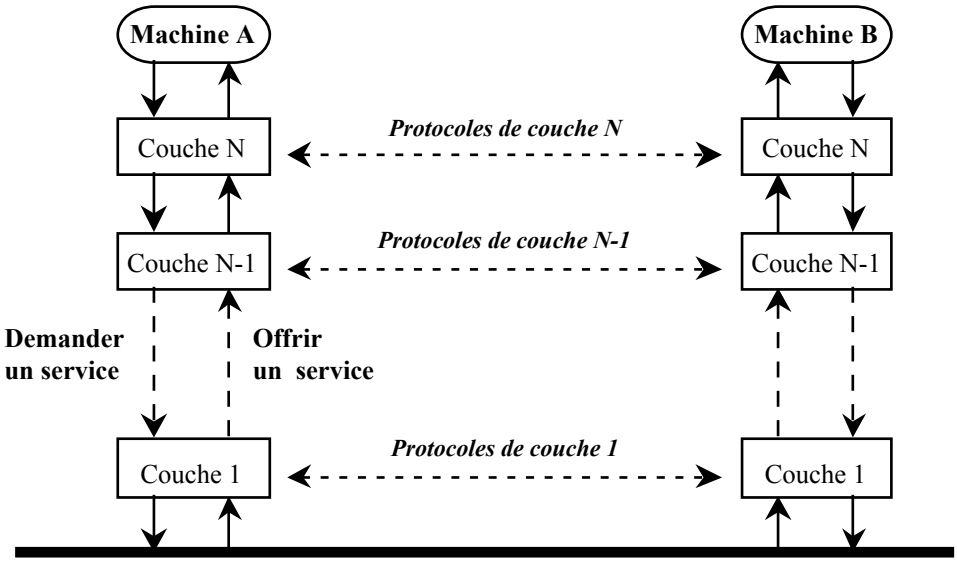
1. Architecture des réseaux, protocoles et services

Pour communiquer sur un réseau, les machines utilisent un ensemble de règles et de conventions appelées **PROTOCOLES**. Partant du principe de modularité et compte tenu de leur complexité, les protocoles ont été structurés **en couches** dans le but de faciliter et de contrôler leur implémentation.

L'un des avantages de cette structuration est d'isoler les différents protocoles pour que tout changement introduit sur l'un d'eux n'affecte pas les fonctionnalités des autres.

Ce modèle offre des interfaces entre les différentes couches afin de permettre aux protocoles d'une couche donnée d'interagir avec ceux des couches qui lui sont directement adjacents. En effet, chacune des couches s'appuie sur des **services** offerts par une couche inférieure et vise à offrir ses services à la couche qui lui est supérieure.

Voici un schéma résumant ces définitions:



2. Modèle OSI

L'**ISO** (International Standards Organization) est un organisme de standardisation qui a défini une architecture normalisée pour les réseaux. Cette architecture est connue sous le nom de modèle **OSI** (Open Systems Interconnection). Ce modèle propose une décomposition en sept couches, chacune réalisant une fonction bien définie et correspondant à un certain niveau d'abstraction.

Les sept couches sont:

- Couche 1: Couche physique;
- Couche 2: Couche liaison de données qui est divisée en deux sous-couches: la sous-couche MAC (Medium Access Control) et la sous couche LLC (Logical Link Control);
- Couche 3: Couche réseau;
- Couche 4: Couche transport;
- Couche 5: Couche session;
- Couche 6: Couche présentation;
- Couche 7: Couche application.

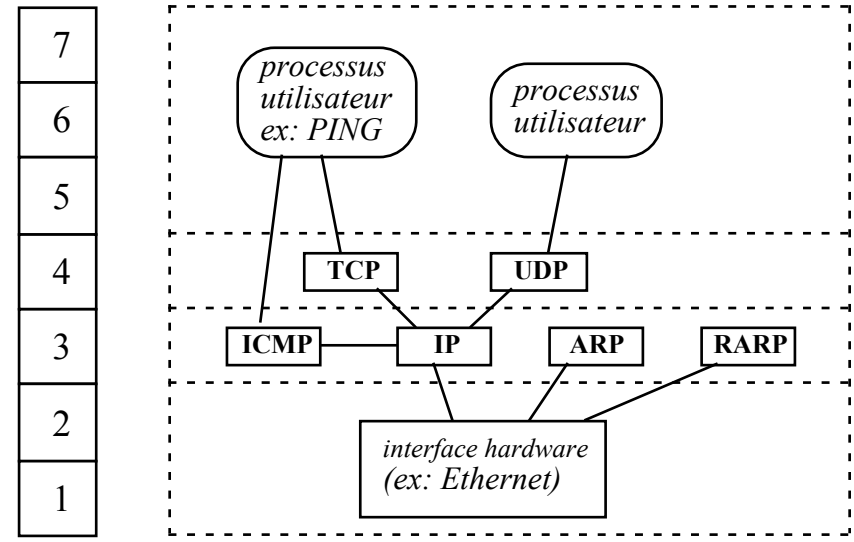
3. Présentation des protocoles étudiés

Dans ce TP, nous aurons à étudier la **famille de protocoles utilisés dans Internet (TCP/IP)** correspondant aux couches 3 et 4. Par famille de protocoles, on entend un ensemble de protocoles appartenant à des couches différentes formant une base minimale permettant la communication sur un réseau.

La famille de protocoles TCP/IP est un exemple de famille de protocoles qui est devenue un standard de fait, de part son utilisation courante dans le monde.
Cette famille contient:

- Les protocoles **ARP/RARP** (Address Resolution Protocol/Reverse Address Resolution Protocol), **IP** (Internet Protocol) et **ICMP** (Internet Control Message Protocol) associés à la couche 3.
- Les protocoles **TCP** (Transmission Control Protocol) et **UDP** (User Datagram Protocol) associés à la couche 4.

Le schéma suivant montre les diverses interactions qui existent entre les protocoles de la famille de protocoles TCP/IP:



Modèle OSI

3-1. Le protocole de la couche 2.

C'est le protocole CSMA/CD (norme ISO 802.3) qui est utilisé dans les cartes Ethernet qui sont sur les stations (voir séance précédente).

3-2. Protocoles de la couche 3

Dans cette section, nous commencerons par définir les protocoles ARP/RARP, ensuite nous vous donnerons un aperçu sur le protocole IP et nous terminerons par une étude sommaire du protocole ICMP.

A- Les Protocoles ARP/RARP:

Le protocole ARP (Address Resolution Protocol) permet à une machine A de trouver, si elle existe, l'adresse Ethernet d'une autre machine B connectée sur le même réseau, en donnant uniquement l'adresse Internet de celle-ci.

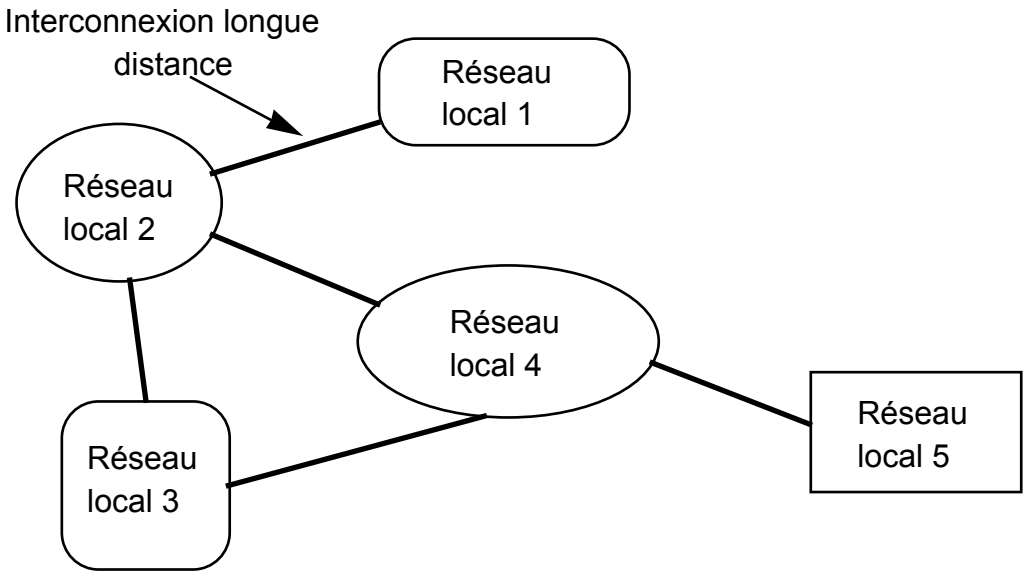
Le but de ce protocole est de cacher aux applications opérant au niveau supérieur l'adresse physique des machines et de ne leur permettre de manipuler que les adresses Internet.

Le protocole RARP (Reverse-ARP) permet de trouver l'adresse INTERNET d'une machine du réseau à partir de son adresse Ethernet (intéressant pour « booter » une machine via le réseau).

L'utilisateur n'a pas directement accès à ces deux protocoles: ils sont automatiquement utilisés par le protocole IP, quand cela est nécessaire.

B- Le Protocole IP:

Pour bien comprendre la problématique résolue par les protocoles de la couche 3, il faut bien avoir à l'esprit la nuance qui existe entre un réseau local (LAN) et un réseau étendu (WAN), les deux étant souvent appelé "réseau". Un réseau local est un réseau regroupant un ensemble de machines peu éloignées géographiquement (par exemple un réseau Ethernet). Un réseau étendu est un ensemble de réseaux locaux interconnectés par des liens « longue distance ».



Un réseau étendu

Ainsi, le protocole Internet permet aux couches supérieures de faire abstraction de l'ensemble des réseaux (locaux) qu'il faut parfois traverser pour acheminer un paquet dans un

réseau (étendu). Ainsi les couches supérieures n'ont pas à se soucier de la route parfois compliquée que les paquets doivent suivre, elles voient la liaison comme une liaison directe entre la machine émettrice et la machine réceptrice. Ce problème de gestion des routes à prendre dans un "réseau étendu" est appelé routage. D'autre part, ces différents réseaux composant le "réseau étendu" peuvent être hétérogènes, c'est-à-dire utiliser des protocoles différents (au niveau 2) et des trames de différentes longueurs. Le découpage et le réassemblage des paquets (appelé mécanisme de fragmentation) est aussi résolu par ce protocole de la couche 3.

Le service de base offert par IP est l'émission et la réception de paquets de données appelés datagrammes. Ce service est dit "non fiable" dans la mesure où la perte (ou l'altération) d'un paquet pendant son transport n'est pas résolue. Aucun mécanisme permettant de récupérer ces erreurs n'existe dans la couche IP.

C- Le Protocole ICMP:

Nous avons vu que IP offre un service non fiable, c'est-à-dire que si un paquet est perdu ou qu'une anomalie quelconque se produit au niveau des fonctionnalités de IP, celui ci ne rapporte aucune information quant à l'occurrence, la nature ou l'origine d'une telle erreur.

Pour remédier à cette faiblesse du protocole, les concepteurs ont introduit dans la famille de protocoles TCP/IP un mécanisme appelé ICMP. La fonctionnalité principale du protocole ICMP est de rapporter, à la station émettrice du paquet, les erreurs qui peuvent se produire au niveau IP.

Ainsi, quand le protocole IP n'arrive pas, dans certains cas, à remplir correctement la tâche qui lui est assignée, il l'indique au protocole ICMP qui émet un paquet à destination de la station source notifiant la nature et l'origine de l'erreur. Ce paquet est récupéré par le protocole ICMP de la station source qui informe le protocole IP de l'occurrence de cette erreur. Le protocole IP ainsi avisé agira en conséquence.

3-3. Protocoles de la couche 4

Dans la famille de protocoles TCP/IP, la couche de niveau 4 est la couche la plus haute avec laquelle les processus communiquent pour envoyer ou recevoir des données. Deux protocoles de niveau 4 sont définis : TCP et UDP. Du point de vue de l'utilisateur, les services proposés par ces deux protocoles sont très différents. Le seul point commun de ces deux protocoles est le fait qu'ils se basent tous les deux sur le protocole IP pour acheminer les données.

A- Le protocole TCP:

Le service offert par TCP peut être comparé à celui offert par le téléphone: Quand vous téléphonez à quelqu'un, le dialogue ne peut s'instaurer qu'a partir du moment ou votre

interlocuteur décroche son combine et dit "ALLO" (ce qui correspond a l'établissement de la connexion): vous pouvez alors dialoguer...

Un protocole offrant un tel service, en l'occurrence TCP, est dit un protocole **orienté connexion**: cela signifie que si deux processus veulent s'échanger des données par TCP, ils doivent préalablement établir une connexion virtuelle. Une fois la connexion établie, TCP garantit que toutes les données envoyées par le premier processus seront reçues sans la moindre erreur par le deuxième: il n'y aura ni perte, ni modification des données. De plus, si les données sont envoyées dans un certain ordre, elles seront réceptionnées dans le même ordre.

Le service proposé par TCP possède aussi la caractéristique d'être de type "**byte-stream**" (flux d'octets): si le premier processus envoie 5 puis 15 caractères, ceux-ci peuvent être récupérés de différentes manières par le processus distant:

- en une lecture de 20 caractères,
- deux lectures de 10 caractères,
- deux lectures de 7 et une lecture de 6 caractères...

Autrement dit, il n'y a pas de découpage fixe dans le flux de données véhiculées.

Notez bien qu'un même processus peut gérer simultanément plusieurs connexions TCP. Il peut par exemple ouvrir deux connexions avec deux processus s'exécutant sur deux machines différentes.

B- Le protocole UDP:

Le service offert par le protocole UDP peut être comparé à celui offert par la poste: quand vous postez une lettre, et si vous demandez le tarif habituel, il peut arriver que cette lettre se perde. Par ailleurs, il se peut qu'elle arrive à destination après une autre lettre qui avait pourtant été postée après elle...

De la même manière, deux processus peuvent aussi utiliser le protocole UDP pour s'envoyer des données. Avec UDP, **aucune connexion préalable n'est nécessaire**, mais à l'inverse de TCP, UDP ne donne aucune garantie quant à la qualité du service proposé: des données peuvent avoir été perdues, arriver dans le désordre, éventuellement avoir été modifiées... C'est à l'utilisateur d'effectuer ces contrôles, si cela est nécessaire.

Alors que TCP véhicule un flux de données entre les processus, UDP véhicule des paquets de données: si un premier processus envoie 5 puis 15 caractères par UDP, le processus distant ne pourra pas les lire en une seule fois: il devra lire les deux paquets séparément.

Avec TCP, l'unité d'information est l'octet; avec UDP c'est le message.

En résumé, deux processus peuvent communiquer en utilisant le protocole TCP ou UDP. Quels peuvent être les critères permettant de choisir l'un plutôt que l'autre ? Les critères

les plus évidents sont directement liés aux caractéristiques de TCP et UDP. Par exemple, si l'on veut absolument une communication fiable à 100%, on choisira TCP...

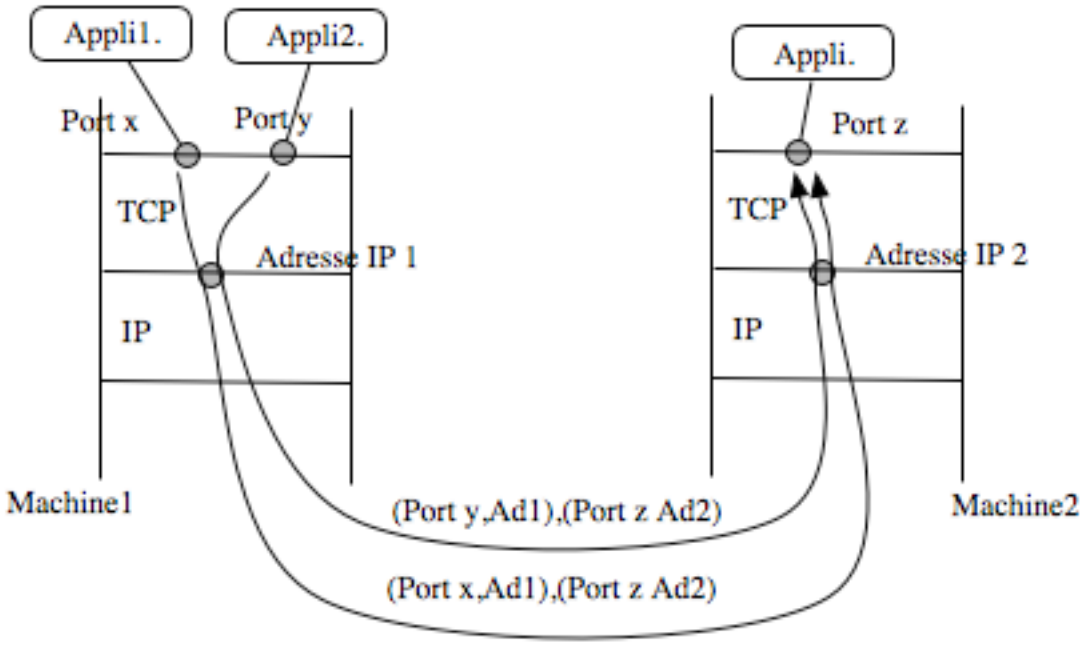
Cependant il faut aussi tenir compte du type de données échangées. Par exemple, un processus client dialoguera plutôt avec un processus serveur en utilisant le protocole UDP si les messages sont de type "**requête**" - "**réponse**". Par contre, transférer un fichier revient a envoyer un flux continu d'octets, et on préférera le protocole TCP pour une telle communication.

C- Problème de l'adressage au niveau 4:

Dans le TP précédent, vous avez vu qu'une machine était identifiée par une adresse INTERNET; C'est cette adresse que TCP indique à IP quand il veut envoyer un message, elle se retrouve ensuite dans l'entête IP. Mais au niveau 4, cette adresse n'est pas suffisante, que vous vouliez utiliser TCP ou UDP.

Prenons le cas du protocole TCP: à un instant donné, on peut imaginer que plusieurs connexions ont été ouvertes par plusieurs processus sur une même machine; l'adresse INTERNET de la machine ne permet pas, à elle seule, de distinguer ces connexions; la notion de port a été introduite pour faire ces distinctions.

Un port est un entier de 16 bits qui sert à identifier un point d'accès aux protocoles de la couche 4. Ainsi une connexion TCP peut être identifiée de façon unique par deux couples [adresse INTERNET, numéro de port]. Quand un processus veut envoyer un paquet vers un autre processus d'une machine distante, il doit indiquer l'adresse INTERNET de cette machine, et un numéro de port particulier sur celle-ci. Voici un schéma qui résume ces notions pour un petit exemple (page suivante) :



Deux connexions TCP

D- Interface logicielle sur les protocoles TCP et UDP:

Jusqu'ici, nous vous avons présenté les aspects théoriques de TCP et UDP. Comme ces deux protocoles sont directement accessibles à n'importe quel processus utilisateur, cela signifie que le système met à la disposition de l'utilisateur un certain nombre de primitives permettant d'envoyer et recevoir des données par TCP ou UDP.

Toutes ces primitives constituent une interface (API: Application Program Interface).

Il existe plusieurs interfaces possibles, selon le type de système d'exploitation et le langage de programmation utilisé. Sous les systèmes UNIX, les deux interfaces les plus utilisées sont:

- **les sockets** BSD (Berkeley Software Distribution).
- Systeme V TLI (Transport Layer Interface).

Dans ce TP, vous utiliserez les **sockets**. Le langage de programmation à utiliser est bien évidemment le C, mais dans un premier temps, vous ne serez pas amenés à développer.

Pour étudier les protocoles TCP et UDP, vous utiliserez un utilitaire vous proposant une interface souple et simplifiée sur les sockets: **socklab.** Ce "laboratoire" à socket vous permet en fait d'appeler de façon interactive les primitives de base de manipulation des sockets. Une socket est un "point d'accès" que vous devez créer pour envoyer ou recevoir des données par le protocole UDP ou TCP.

DEROULEMENT DU TP:

Dans ce TP, au moins 3 stations doivent être connectées sur un même réseau.

Les captures et les analyses des paquets générés par les manipulations se feront à l'aide de l'utilitaire **ethereal** (cf. documentation sur les outils des TP). Ces captures pourront se faire sur l'une quelconques des 3 machines.

1. Etude des protocoles de niveau 3

Dans cette section, nous abordons l'étude des protocoles de la famille TCP/IP.

1-1. Le protocole ARP

1-1-1 Syntaxe des paquets ARP

Le but de la manipulation suivante est de générer et d'analyser des paquets de type ARP. Il arrive parfois, comme vous l'avez constaté dans le premier TP, que les paquets ARP ne soient pas systématiquement générés parce que les correspondances entre adresse Internet et Ethernet sont conservées dans une table que le protocole ARP met à jour périodiquement.

Dans certaines étapes des manipulations suivantes, on utilisera la commande **arp –a -d** pour vider le contenu de cette table.

- Sur une première station, exécutez **arp –a -d**, puis, à l'aide de **ping**, vérifiez la présence d'une seconde station du réseau. Lorsque deux paquets ont été capturés, arrêtez **ethereal.** Retrouvez dans le format hexadécimal, les valeurs des différents champs (entête ETHER et données ARP) des paquets.

- Qu'est-ce qui permet d'identifier les paquets comme étant de type ARP ?
  Indiquez le rôle des différents champs ARP.

Format de la trame Ethernet : (entre parenthèses est indiqué le nombre de bits des champs)

0	8	14	20	22	72-1526
Préambule (64) + marqueur début (8)	Destination @ (48)	Source @ (48)	Type (16)	**Data (368-12000)**	CRC (32)

- Où se situe le paquet ARP dans la trame Ethernet?
  De la même manière, faites le schéma correspondant au paquet ARP.

- Comment le niveau Ethernet connaît-il la taille des paquets qu'il reçoit ? Comment détermine-t-il la fin du paquet ?

- Qu'est-ce qui est transmis à la couche ARP par la couche Ethernet (à la réception d'un paquet) ? Ethernet connaît-il l'existence d'un bourrage à la réception d'un paquet ?

1-1-2 Algorithme du protocole ARP

Le but de cette section est de découvrir l'algorithme général qui implémente le protocole ARP.

Votre tâche consiste à partir des manipulations et des observations détaillées plus loin, à compléter de façon informelle le corps de l'algorithme suivant :

tantque vrai **faire**
attendre(événement**)**
si événement est "Question sur adresse internet" (requête interne de IP vers ARP)
...
finsi
si événement est expiration du timer associé à une entrée
...
finsi
si événement est réception requête (ARP Request)
...
finsi
si événement est réception réponse (ARP Reply)
...
finsi

... (attention il y a d'autres évenements)
fin tantque

Au cours des opérations suivantes, vous serez amenés à manipuler manuellement le contenu des tables du protocole ARP. La commande à utiliser est **arp** (cf doc).

Vous allez réaliser des manipulations sur les trois stations libres de la plate-forme (une station est réservée à la capture des paquets). Dans la suite, nous identifierons ces stations par A, B et C.

Voici la liste des manipulations que vous devez réaliser :

1- Ajoutez dans la table ARP de A, une entrée pour la station B. Réciproquement, ajoutez dans la table de B, une entrée pour la station A.

Exécutez sur A un ping vers B, et notez les paquets engendrés. Dans la table de A, supprimez l'entrée pour la station B.
Recommencez le ping. Notez les paquets engendrés.

2- Dans la table de A, supprimez l'entrée pour la station B.
Dans la table de B et C, supprimez l'entrée pour la station A.
Exécutez sur A un ping vers B, et **AUSSITOT APRES**, consultez le contenu de la table de B et C. Qu'est-ce qui a changé ?

Pendant un moment (au moins 3 minutes) ne touchez plus aux stations B et C, puis consultez leurs tables.
En fait l'effacement dans la table de A de l'adresse de B va prendre à peu près 20 minutes, n'attendez pas.

3- Faites un ping vers une station que vous aurez au préalable débranchée du réseau (et dont l'adresse n'est pas dans la table ARP de la station d'où est fait le ping). Quelle est la durée du timer de réémission d'une requête ARP ? Combien de tentatives sont faites par ARP avant abandon ?

4. Videz les tables de A et B. Dans la table de A, ajoutez une entrée pour l'adresse de B mais en précisant l'option **pub** à la fin de la commande **arp** (**pub** signifie "**published**". Vérifiez que cette entrée est bien Publish quand vous demandez le contenu de la table par **arp -a**.

Observez l'activité sur le réseau au moment où vous rajoutez cette entrée dans la table ARP avec l'option PUB.
Videz la table ARP de C. Exécutez sur C un ping vers B. Notez à nouveau, les paquets engendrés, et le contenu de la table de A, B et C.

Pourquoi deux paquets de type **ARP reply** sont-ils observés ? Expliquez ce qu'il s'est passé (demandez une analyse détaillée de ces paquets; étudiez en particulier les adresses Ethernet et Internet).

Dans quels cas la déclaration d'une adresse en Publish peut-elle être intéressante ?

Que se passe-t-il si deux machines possèdent la même adresse Internet sur un réseau local ? Essayez.

Remarque: vous pouvez réaliser d'autres manipulations, afin de vérifier la validité de vos conclusions.

1-2. Le protocole ICMP

Rappel: **Quand une station rencontre un problème lors de la réception d'un datagramme, elle retourne un message ICMP à la station source pour signaler ce problème. ICMP est aussi utilisé pour tester le réseau.**

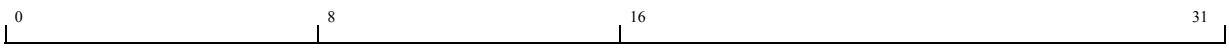
Il existe une douzaine de types de messages ICMP. Chaque type de message est encapsulé dans un paquet IP,voici les types de message les plus couramment utilisés (pour une liste plus détaillée, cf Comer page 127).

- **ECHO REQUEST & ECHO REPLY** sont utilisés pour voir si une destination donnée (une station) est accessible et fonctionne. A la réception d'un message ECHO REQUEST, le destinataire doit répondre par un message ECHO REPLY.
- **DESTINATION UNREACHABLE** est généré quand le protocole IP ne sait pas comment joindre la station à qui est destiné un datagramme. Par exemple quand il n'a pas de réponse du protocole ARP.
- **SOURCE QUENCH** est utilisé pour "brider" les stations qui envoient un trop grand nombre de datagrammes. A la réception de ce message, la station devrait modérer sa cadence d'émission des datagrammes.
- **TIME EXCEEDED** est envoyé lorsqu'un datagramme à son compteur (Time To Live) à zéro et doit donc être détruit.
- **PROBLEM PARAMETER** indique qu'une valeur illégale a été détectée dans un champ de l'entête d'un datagramme.

Le format du header ICMP varie suivant le type de message qu'il véhicule. Nous étudions dans cette section un exemple de messages ICMP qui sont utilisés par la commande ping (ECHO REQUEST et ECHO REPLY).

Bien qu'ils soient différents, les paquets ICMP possèdent tous un premier champ de 16 bits qui identifie le type et le code du message contenu dans le paquet. Le type 8 identifie par exemple un ECHO REQUEST et le type 0 un ECHO REPLY. Dans les deux cas, le code est à 0.

Le format du **paquet ICMP** est le suivant:



Type (8)	Code (8)	Checksum (16)
Identifier (16)		Sequence number
Optional data		

Les champs **IDENTIFIER** et **SEQUENCE NUMBER** permettent uniquement de faire correspondre la réponse à une requête donnée.

Pour envoyer un paquet ICMP, vous aurez à remplir un fichier avec les valeurs correspondantes aux différents champs du paquet ICMP. Ensuite vous utiliserez le logiciel **send_icmp** pour envoyer le paquet:

 # send_icmp <host> <nom de fichier>

où <nom de fichier> est le nom du fichier que vous avez créé et <host> est la station destination.

Dans un fichier, écrivez en décimal ou hexadécimal (dans ce cas ajoutez x avant la valeur) les valeurs des différents octets correspondants au paquet ICMP. Les valeurs de chaque octet doivent être séparées par des blancs ou des passages à la ligne.

***Remarque*:** Le calcul du checksum nécessite que la valeur initiale du champ checksum soit nulle.

Envoyez le paquet à l'aide de **send_icmp** et observez ce qu'il se passe. Si vous n'avez pas reçu de réponse, vérifiez le format de votre paquet et refaites la manipulation.

Demandez une analyse hexadécimale des paquets **request** et **reply**. Que remarquez vous pour le champ Identifier et le champ Séquence Number ?

Donner l'algorithme de calcul des deux octets de Checksum. Quels genres d'erreurs ce type de vérification peut elle détecter et ne pas détecter?

Pour vérifier vos conclusions, refaites la manipulation en construisant des paquets différents.

2 Etude des protocoles de niveau 4

Dans cette section, vous allez étudier les protocoles TCP/IP de niveau 4, à savoir UDP et TCP.

Vous utiliserez ici le logiciel **socklab** (voir documentation fournie), dont le but est de vous proposer une interface sur les primitives de manipulation des sockets.

2-1 Le protocole UDP

Sur deux stations différentes, lancez **socklab** en précisant que vous désirez uniquement travailler avec le protocole UDP:

 # socklab udp

Sur chacune des deux stations, créez une socket UDP. Ces deux sockets seront utilisées pour échanger des messages entre les deux stations:

socklab-udp> sock

Equivalent des commandes en mode standard: **sock udp** (création de socket udp) et **bind** (affectation d'adresse IP et de numéro de port). Voir documentation sur Socklab.

Notez bien le numéro de port qui vous est retourné: il identifie de facon unique la socket sur la machine.

Sur une des deux machines, demandez à émettre un paquet de données vers l'autre machine, en précisant son nom et le numéro de port de la socket précédemment créée:

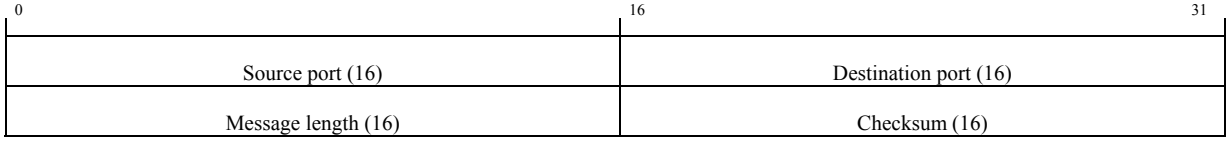
socklab-udp> sendto <Id de Socket> <nom de machine> <numéro de port>

Tapez et validez la chaîne de caractères qui doit être transmise vers la machine distante. A quoi sert l'identificateur de socket ?

Sur la deuxième machine, demandez à recevoir un paquet de données sur la socket précédemment créée, en précisant le nombre maximum d'octets à lire:

socklab-udp> recvfrom <Id de Socket> <nb d'octets>

Capturez et analysez le (ou les) paquets engendrés par l'émission du message précédent. Sachant qu'un paquet UDP est structuré de la façon suivante:



Format de l'entête UDP

- Précisez le rôle de chaque champ de l'entête UDP. Quelles sont les informations passées à IP par UDP à l'émission d'un paquet (données + paramètres de service) ?
- Recommencez les manipulations précédentes (émission et réception de données), en permutant éventuellement le rôle des deux stations (une socket peut servir à la fois pour émettre et recevoir des données).

Effectuez les variantes suivantes:

- Demandez la réception avant l'émission des données;
- Envoyez plusieurs paquets avant de demander leur réception;
- Croisez l'émission des paquets: sur chacune des stations, demandez d'envoyer un paquet, puis demandez de lire le paquet envoyé par la station distante.
- Notez ce qui se passe quand vous demandez à recevoir plus ou moins d'octets que la station distante en envoie.
- Envoyez un paquets vers une machine que vous aurez au préalable, débranchée du réseau. Observez ce qui se passe à l'aide d'une troisième machine.
- Envoyez plusieurs paquets de taille importante vers une machine de façon à "saturer" le récepteur (remplir son buffer de réception). Par exemple envoyer 6 paquets de 5000 octets, puis essayez de les réceptionner (Sous socklab taper **# 5000** à la place du message à envoyer).

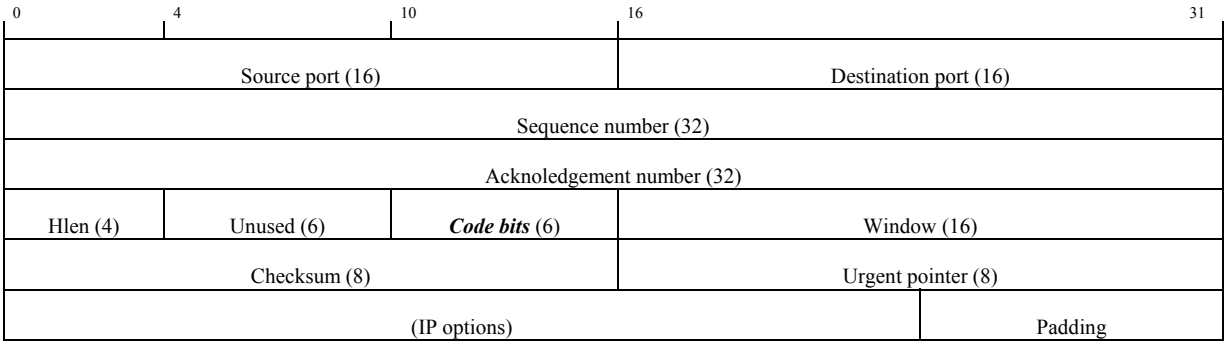
  Vous pouvez connaître la taille du buffer de réception à l'aide de la commande ***options*** de Socklab.
- Envoyez un paquet UDP vers un port inexistant.

Observez ce qui se passe à l'aide d'une troisième machine et analysez les paquets qui sont alors échangés entre les machines. Quel protocole est utilisé alors ? Expliquez ce qui se passe entre les différents protocoles de la couche réseau.

D'après toutes ces observations, écrivez un résumé sur le fonctionnement du protocole UDP.

2-2. Le protocole TCP

Voici l'entête des paquets TCP:



Format de l'entête TCP

Le champ Code bits est composé des six flags suivants : URG, ACK, PSH, RST,SYN, FIN.

Signification des différents champs:

- **SOURCE PORT ET DESTINATION PORT** ont le même sens que dans les paquets UDP.
- **SEQUENCE NUMBER et ACK NUMBER** sont utilisés pour le séquencement et le contrôle d'erreur des données (le flag ACK indique si le champ ACK NUMBER contient une valeur valide).
- **HLEN** indique la taille de l'entête TCP en mots de 4 octets (la taille de l'entête TCP est variable: elle peut être complétée par une ou plusieurs options de 4 octets chacune).
- **CHECKSUM** a le même sens que dans les paquets UDP.
- **URGENT POINTER** est utilisé pour le transport de "données urgentes" (le flag URG indique si le champ URGENT POINTER contient une valeur valide).
- Les flags **SYN et FIN** sont utilisés pour l'établissement et la fermeture des connexions virtuelles.
- Le flag **RST** est utilisé pour signaler au destinataire une demande de re-initialisation des connexions qui sont dans un état incertain (SYN dupliqués, fermeture anormale, panne …).
- Le flag **PSH** ne sera pas étudié.
- Le champ **WINDOW** est utilisé pour le contrôle de flux.

2-2-1 Etablissement d'une connexion

Sur deux machines, lancez **socklab** en précisant que vous désirez uniquement travailler avec le protocole TCP:

 # socklab tcp

- Sur la première machine, créez une socket "**passive**". Notez le numéro de port de la socket ainsi créée, puis mettez-la en attente de connexion.

  socklab-tcp> passive

  Equivalent des commandes en mode standard ***sock tcp*** *(création de socket),* ***bind*** *(affectation d'adresse IP et de numéro de port) et* ***listen*** *(attente) sur cette socket.*

  socklab-tcp> accept

- Sur la deuxième machine, créez une socket "**active**", et connectez-la sur la machine et le port de la socket passive précédemment créée. Ceci peut se faire grâce à la commande:

  socklab-tcp> connect <nom de machine> <numero de port>
  Equivalent des commandes en mode standard: ***sock tcp*** *(création de socket),* ***bind*** *(affectation d'adresse IP et de numéro de port locaux) et* ***connect*** *(requête de connexion) sur cette socket.*

- Pourquoi la première socket créée est dite "**passive**", et la seconde "**active**" ?

- Quels sont les rôles respectifs de ces deux sockets dans l'établissement de la connexion?

- Analysez les paquets générés lors de l'établissement de la connexion.
  - Décomposez les étapes de cette connexion: enchaînement dans le temps des demandes de services à TCP et des messages échangés.
  - Quel est le rôle du flag SYN?
  - Quelles sont les informations échangées durant ces étapes ? A quoi servent les numéros de séquence et d'acquittement ? (attention les numéros donnés par Ethereal sont « relatifs », voir les vrais dans l'hexadécimal)
  - A quoi servent les options de TCP ?

- Expliquez ce qui se passe au moment de l'"accept". Essayez de le faire avant et après le "connect". Regardez la liste des sockets (commande status).

- Ouvrez plusieurs connexions d'une machine vers un même port destinataire.

- Qu'est-ce qui identifie réellement une connexion, c'est-à-dire, comment TCP associe les messages reçus aux différentes connexions en cours?

- Faites une demande de connexion (Connect) vers un port inexistant. Expliquez ce qui se passe.

2-2-2 Libération d'une connexion

- Après ouverture d'une connexion entre deux machines, fermez la connexion :

  socklab-tcp> close

- Analysez les paquets générés lors de la fermeture de la connexion de chaque côté. Décomposez les étapes de cette fermeture. Quel est le rôle du flag FIN ?
- Résumez les échanges de messages en spécifiant la valeur des champs spécifiques à cette phase de la communication.
- Après fermeture d'un seul des deux côtés, essayez de continuer à émettre de l'autre côté (par un write). Que se passe t-il (observez les paquets échangés). Si vous refaites un write que se passe t-il ? pourquoi ?
- Essayez différents scénarios de fermeture à l'aide de la commande **shutdown** (voir documentation Socklab).

Par exemple, réaliser une fermeture en sortie (**shutdown out**) sur une machine, puis une émission de données depuis l'autre.

Peut-on continuer à lire les données envoyées ? Peut-on continuer à faire un write après un **shutdown out**.

Faites de même pour le **shutdown in** et **both**.

- Expliquez les avantages et inconvénients de ces différents types de fermeture (close et shutdown).
- Résumez le fonctionnement de TCP en ce qui concerne l'ouverture et la fermeture de connexion grâce à un automate dont les entrées sont les commandes des sockets (CONNECT, WRITE ...) ou l'arrivée de messages particuliers (DATA, SYN, ACK...), et les sorties sont les envois de messages. Un automate de Mealy semble plus approprié.

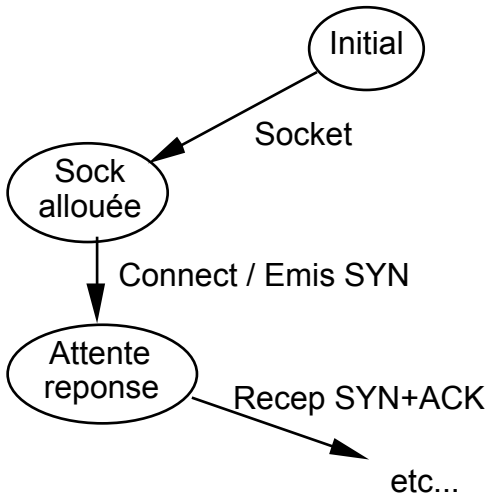


Figure: embryon de l'automate TCP pour une socket active

Faire un automate pour une socket active et un pour une socket dite passive.

2-2-3 Etude du séquencement et du contrôle d'erreur

- Etablissez une connexion par sockets TCP entre deux machines, grâce aux commandes vues précédemment.

- Echangez des données entre les deux machines grâce aux deux commandes **read** et **write** (à la place d'un message normal, vous pouvez utiliser la notation **#nnn** pour envoyer un message de nnn octets).

- Analysez les paquets engendrés par le transport des données. Expliquez le rôle des champs SEQUENCE NUMBER et ACK NUMBER dans l'entête des paquets TCP.

- Donner de façon informelle l'algorithme de mise à jour de ces 2 champs. On fera bien attention à distinguer le cas où l'on reçoit un paquet et le cas ou l'on doit émettre un paquet.

- Envoyez un message vers une machine que vous aurez au préalable, débranchée du réseau. Observez ce qui se passe à l'aide d'une troisième machine. Expliquez en détail les mécanismes utilisés dans ce genre de cas (perte d'un message), comparez avec le protocole UDP.

2-2-4 Contrôle de flux

- Ouvrez une connexion TCP.

- Emettez sur cette connexion un message de taille supérieure à celle du buffer de réception (voir les options de la socket). On peut changer la taille du buffer de réception mais seulement avant l'ouverture de la connexion (côté serveur sur la socket passive).
  Que se passe-t-il ? pourquoi ?
  Faites des read successifs coté récepteur de 5000 octets.

- Analysez les derniers paquets échangés. Regardez en particulier le champ Window de l'entête TCP. Expliquez.

3. Etude de la fragmentation des paquets IP

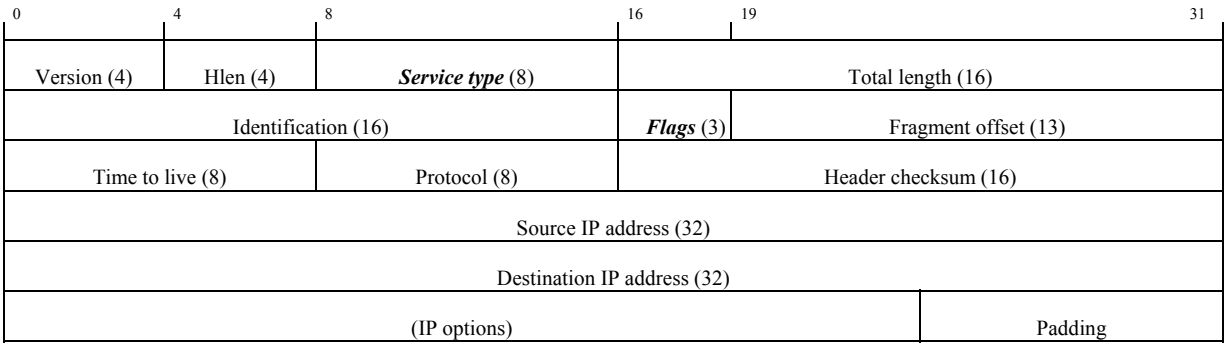
Vous venez d'étudier les principaux protocoles de la famille TCP/IP. Dans cette dernière manipulation, nous vous proposons de revenir sur un aspect particulier du protocole IP: la **fragmentation**.

La fragmentation est un mécanisme géré par le protocole IP lorsque le paquet traité est trop "gros" pour être directement envoyé sur le réseau (par exemple un réseau Ethernet ne peut pas traiter des paquets de taille supérieure à 1518 octets). Dans ce cas, le paquet doit être fragmenté en plusieurs paquets de taille plus petites. Quand un paquet a été fragmenté lors de son émission, il doit évidemment être réassemblé lors de sa réception (toujours au niveau IP).

Les champs **identification, flags et fragment offset** permettent à IP de gérer la fragmentation et le réassemblage.

Remarque : Le flag DF (Don't Fragment) permet d'interdire la fragmentation dans les routeurs intermédiaires. Quand il est positionné à 1, la fragmentation est interdite pour les routeurs suivants. Il se peut alors qu'il soit impossible d'acheminer un paquet. Le paquet est alors détruit et un paquet ICMP est envoyé à la source.

Format de l'entête IP:



Sémantique des champs:

- **VERSION** représente le numéro de version du protocole IP utilisé. Ce champ est nécessaire pour vérifier que l'émetteur et le récepteur et toute machine intermédiaire sont en phase, c'est à dire utilisent le même format de datagramme.
- **HLEN** donne la longueur du header IP en mots de 32 bits. Cette valeur dépend de la longueur des champs IP_OPTION et PADDING qui est variable. SERVICE TYPE est un champ qui apporte des informations sur la façon dont le datagramme doit être traité. Ces informations étant trop précises pour pouvoir être illustrées dans ce TP, nous ne détaillerons pas ce champ. (pour plus de renseignements, cf [Comer] p 93).
- **TOTAL LENGTH** donne la longueur totale du datagramme (en octets), header et données réunis.
- **IDENTIFICATION, FLAGS et FRAGMENT OFFSET**: Ces trois champs interviennent dans le mécanisme de fragmentation qui sera étudié d'une manière détaillée à la fin de ce TP.
- **TIME TO LIVE** représente le temps maximum que doit passer un paquet sur le réseau. Il est décrémenté régulièrement. Une fois le compteur a zéro le paquet est détruit.
- **PROTOCOL** spécifie le type des paquets (TCP, ICMP, UDP) encapsulé dans la trame IP. En réalité il indique le format du champ DATA du datagramme.

- **HEADER CHECKSUM** est un code utilisé pour contrôler l'intégrité du datagramme (c'est-à-dire pour vérifier que ce datagramme n'a subi aucune altération).
- **SOURCE IP ADDRESS et DESTINATION IP ADDRESS** contiennent les adresses internet des stations émettrice et réceptrice du datagramme.
- **IP OPTIONS** est un champ de taille variable (vide par défaut) utilisé généralement pour tester ou débugger un réseau.
- **PADDING** est utilisé pour compléter le champ IP OPTIONS par des bits de bourrage afin que la taille du header soit un multiple de 32 bits.

Expérimentations :

- Sur la station d'observation, lancez **ethereal**.
- Sur une autre station, demandez l'émission d'un paquet UDP de 4600 octets à destination d'un port que vous aurez précédemment créé (voir manipulation précédente).
- Analysez les paquets engendrés sur le réseau. En particulier, étudiez les entêtes IP et expliquez les rôles des champs TOTAL LENGTH, FRAGMENT OFFSET, et du flag MF ("More Fragment"). Répéter si nécessaire la manipulation avec d'autres tailles de paquets.
- Répéter la manipulation en vous servant de TCP. Conclusions?
- Vérifier les conclusions concernant les champs SEQUENCE NUMBER et ACK NUMBER de l'entête de TCP étudiés précédemment dans le cas des 4600 octets de données.

4 Exercices de synthèse (A faire attentivement)

- Observation de la commande talk

Faites un échange de caractères entre deux machines à l'aide de l'utilitaire **talk** (talk <nom utilisateur>@<nom de machine officiel>). Capturez sur une troisième machine les messages échangés lors de ce dialogue.

Analyser les paquets capturés. Enumérez les messages échangés lors de cette manipulation et expliquez ce qu'il se passe au niveau du réseau (protocole d'ouverture de connexion, échange de données et fermeture de connexion).

- Observation de la commande rlogin

Comme précédemment, sur une des machines, capturez les paquets circulant sur le réseau Faites ensuite un ***rlogin*** d'une deuxième machine sur une troisième (compte guest, mot de passe guest). Puis procédez à quelques commandes Unix à travers ce login.

Enumérez les messages échangés lors de cette manipulation et expliquez ce qui se passe au niveau du réseau.

Regardez le contenu du fichier /etc/services.

Références Bibliographiques :

- RESEAUX 4ème Edition
  Andrew Tanenbaum - InterEditions

- Réseaux locaux et Internet (des protocoles à l'interconnexion)
  Laurent Toutain – 2ème Edition - HERMES.

- Analyse struturée des Réseaux
  Des Applications de l'Internet aux infrastructures de télécommunication
  James Kurose et Keith Ross
  2e Edition - Pearson Education

- TCP/IP: Architecture, protocoles, applications
  Douglas COMER, InterEditions, 1992

Très techniques sur les protocoles TCP/IP :

- Principles, Protocols and Architecture - Volume I
  Design Implementation and Internals
  Douglas E. Comer, David L. Stevens

- Internetworking with TCP/IP - Volume II
  Design Implementation and Internals
  Douglas E. Comer, David L. Stevens