REPUBLIQUE TUNISIENNE
*****
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE
*****
DIRECTION GENERALE DES ETUDES TECHNOLOGIQUES
*****
INSTITUT SUPERIEUR DES ETUDES TECHNOLOGIQUES
DE CHARGUIA
*****
Département Technologies de l'Informatique

COURS

ARCHITECTURES ET PROTOCOLES RESEAUX

Réalisé par :

MOHSEN BEN HASSINE

Année Universitaire 2011 - 2012

PLAN DU COURS

FICHE MATIERE (REF : DGET 2009)

Domaine de Formation : Sciences et Technologies	Mention	TI
Mention : Technologies de l'Informatique (TI)	Spécialité	SEM
Parcours : Systèmes Embarqués et Mobiles (SEM)	Semestre	S4
UNITE D'ENSEIGNEMENT (UE) : RESEAUX ET PROTOCOLES	Crédits : 4	Coeff : 4
Volume Horaire par	Semestre : 67.5	Semaine : 4.5

Elément Constitutif (ECUE)	Vol. Horaire / Semestre			Coefficient	Crédits	Evaluation
Architectures et protocoles	Cours	TD	TP	1.5	1.5	Contrôle continu
	11.25	11.25	0			
	22.5					
Ventilation / Semaine	0.75	0.75	0			
	1.5					

PRESENTATION DU COURS

Contexte général :

Dans le cadre de l'unité de l'enseignement Réseaux et protocoles, l'ECUE Architectures et protocoles permet à l'étudiant une connaissance transversale de la pile ARPA ainsi que tous les concepts reliés aux protocoles d'interconnexion et leurs implémentations logicielles.

Objectifs généraux :

1. Comprendre les protocoles ARPA
2. Distinguer les protocoles et les classer par fonction
3. Comprendre l'aspect procédural (algorithmique) de l'implémentation de quelques protocoles

Objectifs spécifiques :

1.1 Distinguer les modèles architecturales liés aux réseaux (OSI, ARPA)
1.2 Analyser les structures de données relatives aux protocoles
2.1 Etudier les concepts simples et avancés relatifs aux communications
2.2 Décortiquer quelques protocoles applicatifs (couche application)
2.3 Enumérer les fonctions et les applications relatifs aux réseaux IP avancés
3.1 Situer les protocoles au niveau architectural (modèle client/serveur, n tiers)
3.2 Comprendre les sockets et explorer leurs aspects algorithmiques

APPROCHE PEDAGOGIQUE

Pré-requis

- Fondements de réseaux
- Algorithmique

Méthodes et outils Pédagogiques

- Cours intégré (CI)
- Exposé informel
- Condensé du cours
- Travaux dirigés
- Coordination CI+TP. (Préparation pour les laboratoires)

Moyens d'évaluation

Effort personnel	20 %
Test + Devoir surveillé	20 %
Examen final	60%

Table des matières

Chapitre 1 : Introduction à la pile ARPA

1 Historique

En 1969 aux Etats Unis, l'agence gouvernementale DARPA lance un projet de réseau expérimental, basé sur la commutation de paquets. Ce réseau, nommé ARPANET, fut construit dans le but d'étudier les technologies de communications, indépendamment de toute contrainte commerciale. Un grand nombre de techniques de communication par modems datent de cette époque.

L'expérience d'ARPANET est alors si concluante que toutes les organisations qui lui sont rattachées l'utilisent quotidiennement pour leurs messages de service. En 1975, le réseau passe officiellement du stade expérimental au stade opérationnel. Le développement d'ARPANET ne s'arrête pas pour autant, les bases des protocoles TCP/IP sont développées à ce moment, donc après qu'ARPANET soit opérationnel. En Juin 1978 Jon Postel définit IPv4 et en 1981 IP est standardisé dans la RFC 791 [J. Postel 1981]. En 1983 les protocoles TCP/IP sont adoptés comme un standard militaire et toutes les machines sur le réseau commencent à l'utiliser. Pour faciliter cette reconversion, la DARPA demande à l'université de Berkeley d'implémenter ces protocoles dans leur version (BSD) d'unix. Ainsi commence le mariage entre ce système d'exploitation et les protocoles TCP/IP.

L'apport de l'Université de Berkeley est majeur, tant au niveau théorique (concept des sockets) qu'au niveau de l'utilisateur, avec des utilitaires très homogènes qui s'intègrent parfaitement au paradigme d'usage existant (rcp, rsh, rlogin. . .). Depuis cette époque, un nouveau terme est apparu pour désigner cette interconnexion de réseaux, l'Internet, avec un " i " majuscule. Le succès de cette technologie est alors très important et suscite un intérêt croissant de la part d'acteurs très divers.

Depuis 1990, ARPANET n'est plus, pourtant le terme Internet demeure il désigne maintenant un espace de communication qui englobe la planète tout entière. Des millions de sites partout sur la surface du globe y sont connectés. Depuis 1994, l'Internet sést ouvert au commerce, surtout avec l'apparition en 1991 d'un nouvel outil de consultation, le " World Wide Web " ou " Web " et ses interfaces populaires : Explorer, Mosaic, Netscape, Mozilla, Firefox, Konqueror. . . La liste n'est pas exhaustive !

Depuis 1995, pour faire face à sa popularité fortement croissante et aux demandes de transactions sécurisées, le protocole évolue et une nouvelle version, la version 6 (IPng puis tout simplement IPv6), est définie et en cours de déploiement expérimental.

Les protocoles désignés par TCP/IP ont également envahi les réseaux locaux eux-mêmes, car il est plus facile d'utiliser les mêmes protocoles en interne et en externe. Pour les utilisateurs, l'accès à l'Internet est possible à l'aide d'une collection de programmes spécialisés si faciles à utiliser que l'on peut ignorer tout (ou presque) de leur fonctionnement interne. Seuls les programmeurs d'applications réseaux et les administrateurs de systèmes ont besoin d'en connaître les arcanes.

Les services réseaux les plus populaires sont principalement :

– Le courrier électronique qui permet l'échange de messages entre usagers.

– Les innombrables forums de discussion (" news ").

– Le transfert de fichiers entre machines (" ftp " et ses dérivés comme " fetch ", " wget ", " curl ". . .).

– Le "remote login ", ou ses équivalents cryptés (" ssh ", qui permet à un utilisateur de se connecter sur un site distant, depuis son poste local.

– Puis maintenant la radio, la vidéoconférence, la réalité virtuelle avec le VRML, le " chat ", les bourses d'échanges point à point, les " blogs " forme évoluée des pages personnelles, les réseaux sociaux,…etc .

En conclusion de ce paragraphe sur l'historique on peut dire que l'Internet est une collection apparemment anarchique (il n'y a pas de structure hiérarchique et centralisée) de réseaux interconnectés et appartenant à divers propriétaires.
On distingue trois niveaux : les réseaux au sein des organisations (lans), les réseaux régionaux et les réseaux de transit.

2 Caractéristiques de TCP/IP

Le succès de TCP/IP, s'il vient d'abord d'un choix du gouvernement américain, s'appuie ensuite sur des caractéristiques intéressantes :

a) C'est un protocole ouvert, les sources (en C) en sont disponibles gratuitement et ont été développés indépendamment d'une architecture particulière ou d'un système d'exploitation particulier ou d'une structure commerciale propriétaire. Ils sont donc théoriquement transportables sur n'importe quel type de plate-forme, ce qui est prouvé de nos jours.

b) Ce protocole est indépendant du support physique du réseau. Cela permet à TCP/IP d'être véhiculé par des supports et des technologies aussi différents qu'une ligne série, un câble coaxial Ethernet, une liaison louée, un réseau token-ring, une liaison radio (satellites, " wireless " 802.11a/b/g), une liaison FDDI 600Mbits, une liaison par rayon laser, infrarouge, xDSL, ATM, fibre optique, la liste des supports et des technologies n'est pas exhaustive. . .

c) Le mode d'adressage est commun à tous les utilisateurs de TCP/IP quelle que soit la plate-forme qui l'utilise. Si l'unicité de l'adresse est respectée, les communications aboutissent même si les hôtes sont aux antipodes.

d) Les protocoles de hauts niveaux sont standardisés ce qui permet des développements largement répandus et inter-opérables sur tous types de machines.

Les majeures parties des informations relatives à ces protocoles sont publiées dans les RFCs (Requests For Comments). Les RFCs contiennent les dernières versions des spécifications de tous les protocoles TCP/IP, ainsi que bien d'autres informations comme des propositions d'améliorations des outils actuels, la description de nouveaux protocoles, des commentaires sur la gestion des réseaux, la liste n'est pas exhaustive.

3 Comparaison TCP/IP — ISO

La suite de protocoles désignée par TCP/IP, ou encore " pile ARPA ", est construite sur un modèle en couches moins complet que la proposition de l'ISO. Quatre couches sont suffisantes pour définir l'architecture de ce protocole.

- Couche Application (Application layer).
- Couche Transport (Transport layer).
- Couche Internet (Internet layer).
- Couche interface réseau (Network access layer).

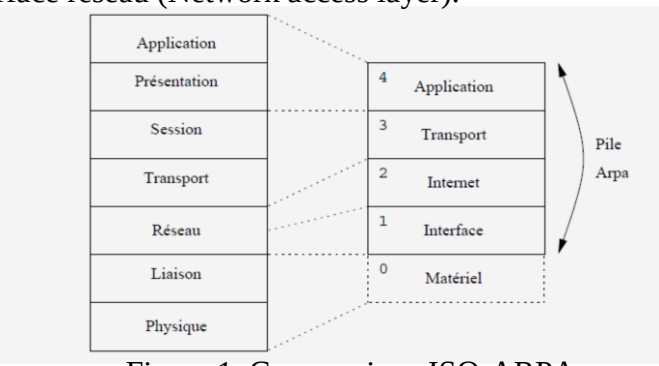


Figure 1. Comparaison ISO-ARPA

La figure 1 met en comparaison les fonctionnalités des couches du modèle OSI et celles des protocoles TCP/IP.

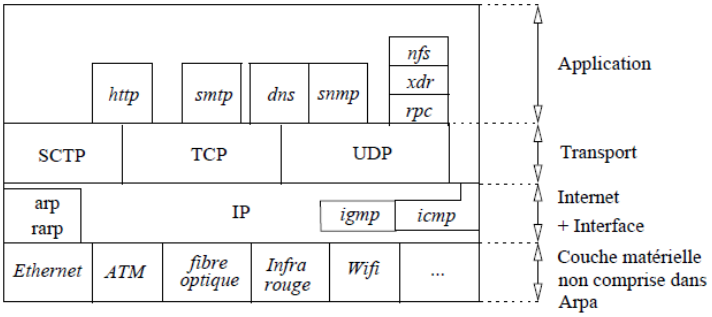


Figure 2. Architecture logicielle

La figure 2 elle, donne une vue d'ensemble de l'architecture logicielle avec quelques protocoles d'applications de la famille IP. Ils sont très nombreux, non représentés tous ici, et il s'en faut de beaucoup car il en existe des centaines. La lecture du fichier /etc/services, présent sur toute machine de la famille des Unix(Linux) , donne un aperçu des principaux services enregistrés auprès de l'IANA.

Légende :

IP " Internet Protocol "
SCTP " Stream Control Transmission Protocol "
TCP " Transmission Control Protocol "
UDP " User Datagram Protocol

3.1 Couche " Application Layer "

Au plus haut niveau les utilisateurs invoquent les programmes qui permettent l'accès au réseau. Chaque programme d'application interagit avec la couche de transport pour envoyer ou recevoir des données. En fonction des caractéristiques de l'échange le programme choisit un mode de transmission à la couche de transport. La plus grande proportion des applications laissent à la couche de transport le soin d'effectuer le travail de " Session ", néanmoins il est possible pour certaines applications de court-circuiter cette fonctionnalité pour agir directement au niveau " Réseau ", comme on peut l'observer sur la figure 2 à droite.

3.2 Couche " Transport Layer "

La principale tâche de la couche de transport est d'assurer la communication entre un programme d'application et un autre. Une telle communication est souvent qualifiée de " point à point ". Cette couche peut avoir à réguler le flot de données et à assurer la fiabilité du transfert : les octets reçus doivent être identiques aux octets envoyés. C'est pourquoi cette couche doit gérer des " checksums " et savoir réémettre des paquets mal arrivés.

Cette couche divise le flux de données en paquets (terminologie de l'ISO) et passe chacun avec une adresse de destination au niveau inférieur. De plus, et c'est surtout valable pour les systèmes d'exploitation multitâches multiutilisateurs (Unix,. . .), de multiples processus appartenant à des utilisateurs différents et pour des programmes d'applications différents, accèdent au réseau au même moment, ce qui implique la capacité de multiplexer et de démultiplexer les données, suivant qu'elles vont vers les réseaux ou vers les applications (" Session ").

3.3 Couche " Internet Layer "

Cette couche reçoit des data-grammes en provenance de la couche interface réseau, qu'elle doit analyser pour déterminer s'ils lui sont adressés ou pas. Dans le premier cas elle doit " décapsuler " son en-tête du data-gramme pour transmettre les données à la couche de transport et au bon protocole de cette couche (TCP, UDP...), dans le deuxième cas elle les ignore. Cette couche prend aussi en charge la communication de machine à machine. Elle accepte des requêtes venant de la couche de transport avec une identification de la machine vers laquelle le paquet doit être envoyé. Elle utilise alors l'algorithme de routage pour décider si le paquet doit être envoyé vers une passerelle ou vers une machine directement accessible. Enfin cette couche gère les datagrammes des protocoles ICMP et IGMP.

3.4 Couche " Network Access "

Le protocole dans cette couche définit le moyen pour un système de délivrer l'information à un autre système physiquement relié. Il définit comment les data-grammes IP sont transmis. La définition de ceux-ci reste indépendante de la couche réseau, ce qui leur permet de s'adapter à chaque nouvelle technologie au fur et à mesure de leur apparition.

4 Encapsulation d'IP

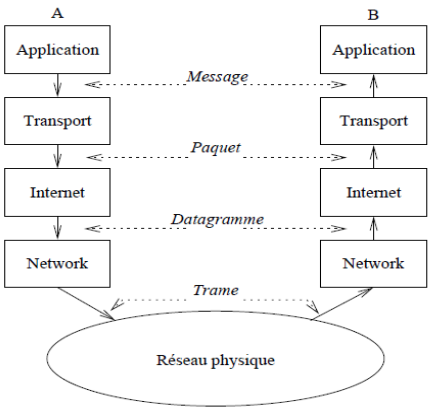


Figure. 3 Encapsulation d'IP

Comme nous l'avons décrit avec le modèle des couches OSI, les couches IP fonctionnent par encapsulations progressives. Chaque couche encapsule la précédente avec les informations de contrôle qu'elle destine à la couche de même niveau sur la machine distante. Cet ajout est nommé " header " (en-tête) parce qu'il est placé en tête des données à transmettre (Figure. 3 et 4)

La taille des " headers " dépend des protocoles utilisés. Pour la couche IP le protocole comporte en standard 5 mots de 32 bits, même chose pour la couche TCP.

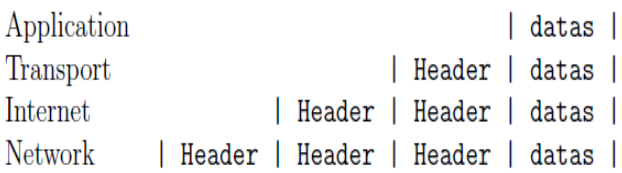


Figure. 4 Les Headers

Chapitre 2 : Les protocoles IP, TCP et UDP

1. Anatomie d'une adresse IP

Une adresse IP est un nombre de 32 bits pour IP_4 et 128 bits pour IP_6 , que l'on a coutume de représenter sous forme de quatre entiers de huit bits (IP_4) , séparés par des points, ou 8 groupes de 2 octets (soit 16 bits par groupe) qui sont séparés par un signe deux-points (IP_6). Exemple :

Adresse IP_4 : 172.31.128.1

Adresse IP_6 : 2001:db8:0:85a3:0:0:ac1f:8001 (présentation en hexa)

Dans la suite on va traiter le cas d'IP_4.

La partie réseau de l'adresse IP4 vient toujours en tête, la partie hôte est donc toujours en queue. L'intérêt de cette représentation est immédiat quand on sait que la partie réseau et donc la partie hôte sont presque toujours codées sur un nombre entier d'octets. Ainsi, on a principalement les trois formes suivantes :

Classe A : Un octet réseau, trois octets d'hôtes.

Classe B : Deux octets réseau, deux octets d'hôtes.

Classe C, D et E : Trois octets réseau, un octet d'hôte.

Classe	Nombre de réseaux/machines
A	1.x.y.z à 127.x.y.z 127 réseaux 16 777 216 machines (2^24)
B	128.0.x.y à 191.255.x.y 16 384 réseaux (2^14) 65536 machines (2^16)
C	192.0.0.z à 223.255.255.z 2 097 152 réseaux (2^21) 256 machines (2^8)
D	224.0.0.0 à 239.255.255.255
E	240.0.0.0 à 247.255.255.255

Figure. 1 Décomposition en classes

Pour distinguer les classes A, B, C, D et E (fig. 1) il faut examiner les bits de poids fort de l'octet de poids fort :

Si le premier bit est 0, l'adresse est de classe A. On dispose de 7 bits pour identifier le réseau et de 24 bits pour identifier l'hôte. On a donc les réseaux de 1 à 127 et 2^{24} hôtes possibles, c'est à dire 16 777 216 machines différentes (de 0 à 16 777 215). Il faut noter que le réseau 0 n'est pas utilisé, il a une signification particulière (" tous les réseaux "). De même, la machine 0 n'est pas utilisée, tout comme la machine ayant le plus fort numéro dans le réseau (tous les bits de la partie hôte à 1, ici 16 777 215), ce qui réduit de deux unités le nombre des machines nommables. Il reste donc seulement 16 777 214 machines adressables dans une classe A!

Si les deux premiers bits sont 10 , l'adresse est de classe B. Il reste 14 bits pour identifier le réseau et 16 bits pour identifier la machine. Ce qui fait 2^{14} = 16 384 réseaux (128.0 à 191.255) et 65 534 (65 536 - 2) machines.

Si les trois premiers bits sont 110 , l'adresse est de classe C. Il reste 21 bits pour identifier le réseau et 8 bits pour identifier la machine. Ce qui fait 2^{21} = 2 097 152 réseaux (de 192.0.0 à 223.255.255) et 254 (256 − 2) machines.

Si les quatre premiers bits de l'adresse sont 1110 , il s'agit d'une classe d'adressage spéciale, la classe D. Cette classe est prévue pour faire du " multicast ", ou multipoint. (RFC

1112), contrairement aux trois premières classes qui sont dédiées à l' unicast ou point à point. Ces adresses forment une catégorie à part.

Si les quatre premiers bits de l'adresse sont 1111 , il s'agit d'une classe expérimentale, la classe E. La RFC 1700 précise " Class E addresses are reserved for future use "

2 Sous-réseaux

En 1984 un troisième niveau de hiérarchie est mis en place : le " subnet " ou sous-réseau, pour permettre aux administrateurs de gérer plus finement de grands réseaux. La RFC 950 donne plus de précisions.

Dans la figure2, les bits 6 et 7 de la partie " host " sont utilisés pour caractériser un sous-réseau. Le " subnet " utilise les bits de poids fort de la partie hôte de l'adresse IP, pour désigner un réseau. Le nombre de bits employés est laissé à l'initiative de l'administrateur.

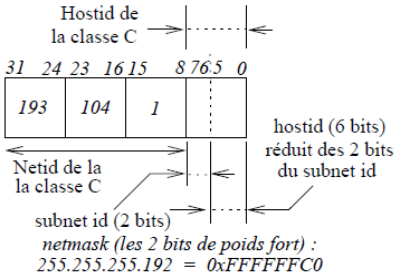


Figure .2 Le subnet

Nous avons d'une part $2^7 + 2^6 = 192$, et d'autre part $2^5 + 2^4 + 2^3 + 2^2 + 2 + 1 = 63$. Ce qui permet de caractériser 4 sous-réseaux de 62 machines (63 moins l'adresse de broascast, le " 0 " n'étant pas compté). Le calcul des masques et des adresses de diffusion est expliqué dans le tableau suivant :

Numéro du réseau	" Netmask "	" Broadcast "	Adressage hôte
193.104.1.00	255.255.255.192	$00 + 63 = 63$	.1 à .62
193.104.1.64	255.255.255.192	$64 + 63 = 127$	.65 à .126
193.104.1.128	255.255.255.192	$128 + 63 = 191$	.129 à .190
193.104.1.192	255.255.255.192	$192 + 63 = 255$	.193 à .254

Soit un total de 62 × 4 = 248 hôtes possibles pour cette classe C avec un masque de sous-réseau, au lieu des 254 hôtes sans.

La machine d'adresse 1 sur chaque sous-réseau, aura comme adresse IP :

Sous-réseau	Adresse	Décomposition
00	193.104.1.1	$00 + 1 = 1$
01	193.104.1.65	$64 + 1 = 65$
10	193.104.1.129	$128 + 1 = 129$
11	193.104.1.193	$192 + 1 = 193$

NB : La notation CIDR du réseau est : 193.104.1.0/26, 26 ➔ 26 bits à 1 pour le subnet mask

3 Le protocole IP

a) Le datagramme IP

IP est l'acronyme de " Internet Protocol ", il est défini dans la RFC 791 et a été conçu en 1980 pour remplacer NCP (" Network Control Protocol "), le protocole de l'Arpanet. Presque trente ans après sa première implémentation, ses limitations se font de plus en plus pénalisantes pour les nouveaux usages sur les réseaux. Avant de le jeter aux orties, posons-nous la question de qui pouvait prévoir à cette époque où moins de mille ordinateurs étaient reliés ensembles, que trois décennies plus tard des dizaines de millions d'hôtes l'utiliseraient comme principal protocole de communication ? Sa longévité est donc remarquable et il convient de l'analyser de près avant de pouvoir le critiquer de manière constructive.

Les octets issus de la couche de transport et encapsulés à l'aide d'un en-tête IP avant d'être propagés vers la couche réseau (Ethernet par exemple), sont collectivement nommés " datagramme IP ", datagramme Internet ou datagramme tout court. Ces datagrammes ont une taille maximale liée aux caractéristiques de propagation du support physique, c'est le " Maximum Transfer Unit " ou MTU.

Quelques caractéristiques en vrac du protocole IP :

– IP est le support de travail des protocoles de la couche de transport, UDP, TCP et SCTP.

– IP ne donne aucune garantie quant au bon acheminement des données qu'il envoie. Il n'entretient aucun dialogue avec une autre couche IP distante, on dit aussi qu'il délivre les datagrammes " au mieux ".

– Chaque datagramme est géré indépendamment des autres datagrammes même au sein du transfert des octets d'un même fichier. Cela signifie que les datagrammes peuvent être mélangés, dupliqués, perdus ou altérés !

Ces problèmes ne sont pas détectés par IP et donc il ne peut en informer la couche de transport.

– Les octets sont lus et transmis au réseau en respectant le " Network Byte Order " ou NBO quelle que soit l'architecture cpu de l'hôte.

– L'en-tête IP minimale fait 5 mots de 4 octets, soit 20 octets. S'il y a des options la taille maximale peut atteindre 60 octets.

b) Description de l'entête

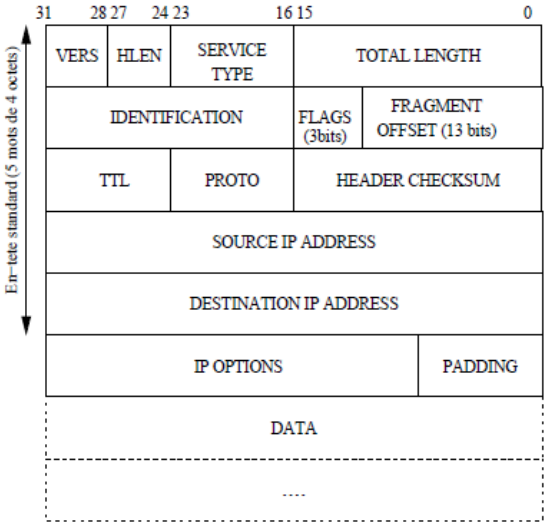


Figure. 3 Le datagramme

- *VERS* : 4 bits qui spécifient la version du protocole IP. L'objet de ce champ est la vérification que l'émetteur et le destinataire des datagrammes sont bien en phases avec la même version. Actuellement c'est la version 4 qui est principalement utilisé sur l'Internet, bien que quelques implémentations de la version 6 existent et soient déjà en exploitation .
- *HLEN* : 4 bits qui donnent la longueur de l'en-tête en mots de 4 octets. La taille standard de cette en-tête fait 5 mots, la taille maximale fait : $(2^3 + 2^2 + 2^1 + 2^0) \times 4 = 60$ octets
- *TOTAL LENGTH* : Donne la taille du datagramme, en-tête plus données. S'il y'a une fragmentation il s'agit également de la taille du fragment (chaque datagramme est indépendant des autres).

La taille des données est donc à calculer par soustraction de la taille de l'en-tête. 16 bits autorisent la valeur 65535. La limitation vient le plus souvent du support physique (MTU) qui impose une taille plus petite, sauf sur les liaisons de type " hyperchannel ".

- *TYPE OF SERVICE* : Historiquement dans la RFC 791 ce champ est nommé TYPE OF SERVICE et joue potentiellement deux rôles selon les bits examinés (préséance « priorité » et type de service). Pratiquement, la préséance ne sert plus et la RFC 1349 définit 4 bits utiles sur les huit (de 3 à 6). Ceux-ci indiquent au routeur l'attitude à avoir vis à vis du datagramme.
- *IDENTIFICATION, FLAGS et FRAGMENT OFFSET* : Ces mots sont prévus pour contrôler la fragmentation des datagrammes. Les données sont fragmentées car les datagrammes peuvent avoir à traverser des réseaux avec des MTU plus petits que celui du premier support physique employé.
- *TTL " Time To Live "* : 8 bits, 255 secondes comme maximum de temps de vie pour un datagramme sur le net. Prévu à l'origine pour décompter un temps, ce champ n'est qu'un compteur décrémenté d'une unité à chaque passage dans un routeur. Couramment la valeur de départ est 32 ou même 64. Son objet est d'éviter la présence de paquets fantômes circulant indéfiniment. Si un routeur passe le compteur à zéro avant délivrance du datagramme, un message d'erreur—ICMP est renvoyé à l'émetteur avec l'indication du routeur. Le paquet en lui-même est perdu.
- *PROTOCOL :* 8 bits pour identifier le format et le contenu des données, un peu comme le champ " type " d'une trame Ethernet. Il permet à IP d'adresser les données extraites à l'une ou l'autre des couches de transport.
- *HEADER CHECKSUM* : 16 bits pour s'assurer de l'intégrité de l'en-tête. Lors du calcul de ce " checksum " ce champ est à 0. A la réception de chaque paquet, la couche calcule cette valeur, si elle ne correspond pas à celle trouvée dans l'en-tête le datagramme est oublié (" discard ") sans message d'erreur.
- *SOURCE ADDRESS,* DESTINATION ADDRESS : Adresse IP de l'émetteur (destinataire), à l'origine du datagramme.
- *IP OPTIONS* : 24 bits pour préciser des options de comportement des couches IP traversées et destinatrices. Les options les plus courantes concernent :
  – Des problèmes de sécurité
  – Des enregistrements de routes
  – Des enregistrements d'heure
  – Des spécifications de route à suivre
  – . . .
- *PADDING* : Remplissage pour aligner sur 32 bits. . .

c) Routage IP

Les datagrammes ne sont pas routés par les machines hôtes, mais par des routeurs dont c'est la fonction par définition. Ils sont plus efficaces et plus perfectionnés pour cette tâche par construction, et surtout autorisent l'application d'une politique de routage (" routing policy ") ce que la pile IP standard d'une machine ne sait pas faire. Toutefois il est courant dans les " petits réseaux ", ou quand le problème à résoudre reste simple, de faire appel à une machine Unix pour ce faire.

La différence entre un " routeur " et un " hôte " est que le premier est capable de transmettre un datagramme d'un interface à un autre et pas le deuxième. Cette opération est délicate si les machines qui doivent dialoguer sont connectées à de multiples réseaux physiques. D'un point de vue idéal établir une route pour des datagrammes devrait tenir compte d'éléments comme la charge du réseau, la taille des datagrammes, le type de service demandé, les délais de propagation, l'état des liaisons, le trajet le plus court. . . La pratique est plus rudimentaire.

On divise le routage en deux grandes familles :

- Le routage direct : Il s'agit de délivrer un datagramme à une machine raccordée au même LAN. L'émetteur trouve l'adresse physique du correspondant (ARP), encapsule le datagramme dans une trame et l'envoie.
- Le routage indirect : Le destinataire n'est pas sur le même LAN comme précédemment. Il est absolument nécessaire de franchir une passerelle connue d'avance ou d'employer un chemin par défaut. En effet, toutes les machines à atteindre ne sont pas forcément sur le même réseau physique. C'est le cas le plus courant, par exemple sur l'Internet qui regroupe des centaines de milliers de réseaux différents.

Sous Unix toutes les opérations de routage se font grâce à une table, dite " table de routage ", qui se trouve dans le noyau lui-même. La figure 4 résume la situation :
Cette table est très fréquemment utilisée par IP : sur un serveur plusieurs centaines de fois par secondes.

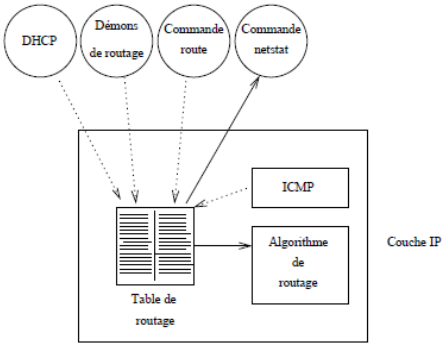


Figure. 4 Table de routage

L'algorithme de routage permet de choisir une destination, en fonction de sa table de routage. Cette opération est essentiellement basée sur le numéro de réseau, I_N, extrait de l'adresse IP (I_D), I_M désigne la machine sur laquelle s'effectue le routage. Pour simplifier voici un pseudo code de l'opération de routage

Pseudo code :

Si I_N est un numéro de réseau auquel M est directement relié alors :

– Obtenir l'adresse physique (MAC) de la machine destinatrice
– Encapsuler le datagramme dans une trame physique et l'envoyer directement.

Sinon Si I_D apparait comme une machine à laquelle une route spéciale est attribuée, router le datagramme en fonction.
Sinon Si I_N apparait dans la table de routage, router le datagramme en fonction.
Sinon S'il existe une route par défaut router le datagramme vers la passerelle ainsi désignée.
Sinon Déclarer une erreur de routage (ICMP).

Le présent algorithme décrit l'approche statique local de la recherche d'une route, Si la topologie d'un réseau offre la possibilité de plusieurs routes pour atteindre une même destination, s'il est vaste et complexe, sujet à des changements fréquents de configuration. Le routage dynamique est alors un bon moyen d'entretenir les tables de routages et de manière automatique (voit les algos : RIP, OSPF, EGP, BGP, GGP)

4 Le protocole TCP

TCP est l'acronyme de " Transmission Control Protocol ", il est défini dans la RFC 793. Les données encapsulées dans un en-tête TCP sont des " paquets TCP ".
Cinq points principaux caractérisent ce protocole :
1. TCP contient un mécanisme pour assurer le bon acheminement des données. Cette possibilité est absolument indispensable dés lors que les applications doivent transmettre de gros volumes de données et de façon fiable.
2. Le protocole TCP permet l'établissement d'un circuit virtuel entre les deux points qui échangent de l'information. On dit aussi que TCP fonctionne en mode connecté (par opposition à UDP qui est en mode non connecté ou encore mode datagramme).
3. TCP a la capacité de mémoriser des données :
– Aux deux extrémités du circuit virtuel, les applications s'envoient des volumes de données absolument quelconques, allant de 0 octet à des centaines (ou plus) de Mo.
– A la réception, le protocole délivre les octets exactement comme ils ont été envoyés.
– Le protocole est libre de fragmenter le flux de données en paquets de tailles adaptées aux réseaux traversés. Il lui incombe cependant d'effectuer le réassemblage et donc de stocker temporairement les fragments avant de les présenter dans le bon ordre à l'application.
4. TCP est indépendant vis à vis des données transportées, c'est un flux d'octets non structuré sur lequel il n'agit pas.
5. TCP simule une connexion en " full duplex ". Pour chacune des deux applications en connexion par un circuit virtuel, l'opération qui consiste à lire des données peut s'effectuer indépendamment de celle qui consiste à en écrire. Le protocole autorise la clôture du flot dans une direction tandis que l'autre continue à être active. Le circuit virtuel est rompu quand les deux parties ont clos le flux.

La figure suivante (fig. 5) montre la structure d'un en-tête TCP. Sa taille normale est de 20 octets, à moins que des options soient présentes

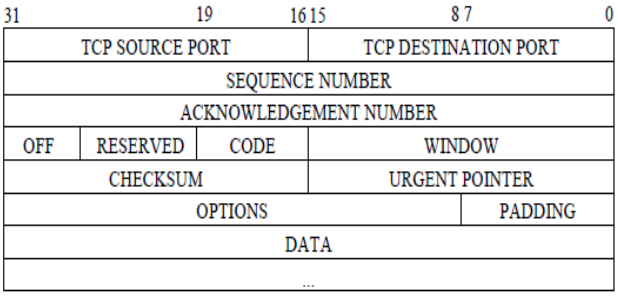


Figure. 5 Structure de l'en-tête TCP

SEQUENCE NUMBER : N° du paquet en cours de transmission
ACK. NUMBER : N° du paquet acquitté jusqu'à lors
OFF : Le champ Offset est codé sur 4 bits et définit le nombre de mots de 32 bits dans l'entête TCP. Ce champ indique donc où les données commencent.
RESERVED : servira pour des besoins futurs
CODE : Six bits pour influer sur le comportement de TCP en caractérisant l'usage du segment

URG Le champ " URGENT POINTER " doit être exploité.
ACK Le champ " ACNOWLEDGMENT NUMBER " doit être exploité.
RST Réinitialisation de la connexion
SYN Le champ " SEQUENCE NUMBER " contient la valeur de début de connexion.
FIN L'émetteur du segment a fini d'émettre.

WINDOW : Le flux TCP est contrôlé de part et d'autre pour les octets compris dans une zone bien délimitée et nommée " fenêtre ". La taille de celle-ci est définie par un entier non signé de 16 bits
URGENT POINTER : Ce champ n'est valide que si le drapeau URG est armé. Ce pointeur contient alors un offset à ajouter à la valeur de SEQUENCE NUMBER du segment en cours pour délimiter la zone des données urgentes à transmettre à l'application.

OPTIONS : C'est un paramétrage de TCP.

L'établissement d'une connexion TCP s'effectue en trois temps, comme le schéma de la *figure 6* l'explicite.

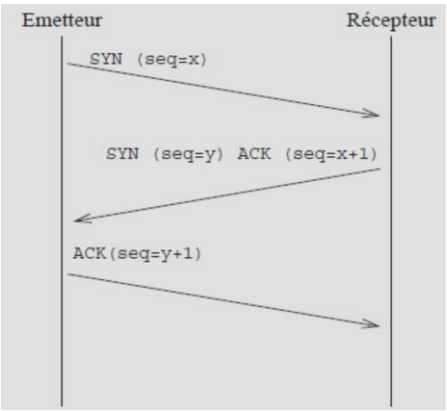


Figure. 6 Etablissement d'une connexion

On suppose que l'émetteur du premier paquet avec le bit SYN a connaissance du couple (adresse IP du récepteur, numéro de port du service souhaité). L'émetteur du premier paquet est à l'origine de l'établissement du circuit virtuel, c'est une attitude généralement qualifiée de " cliente ". On dit aussi que le client effectue une " ouverture active " (*active open*). Le récepteur du premier paquet accepte l'établissement de la connexion, ce qui suppose qu'il était prêt à le faire avant que la partie cliente en prenne l'initiative. C'est une attitude de " serveur ". On dit aussi que le serveur effectue une " ouverture passive " (*passive open*).

1. Le client envoie un segment comportant le drapeau SYN, avec sa séquence initiale (*ISN* = *x*).
2. Le serveur répond avec sa propre séquence (*ISN* = *y*), mais il doit également acquitter le paquet précédent, ce qu'il fait avec ACK (*seq* =*x* + 1).
3. Le client doit acquitter le deuxième segment avec ACK (*seq* = *y* + 1). Une fois achevée cette phase nommée " three-way handshake ", les deux applications sont en mesure d'échanger les octets qui justifient l'établissement de la connexion.

Le bon acheminement des données applicatives est assuré par un mécanisme d'acquittement des paquets. On distingue essentiellement deux manières de faire :

A. Acquittement à fur et a mesure

– Au départ du *Paquet i* une horloge se déclenche. Si cette horloge dépasse une valeur limite avant réception de l'ACK le *Paquet i* est retransmis. Cette valeur limite est basée sur la constante MSL (Max. Segment Lifetime) qui est un choix d'implémentation, généralement de 30 secondes à 2 minutes. Le temps maximum d'attente est donc de 2 ×*MSL*.

– Le temps qui s'écoule entre l'émission d'un paquet et la réception de son acquittement est le RTT (Round Trip Time), il doit donc être inférieur à 2 ×*MSL*. Il est courant sur l'Internet actuel d'avoir un RTT de l'ordre de la seconde.

– L'émetteur conserve la trace du *Paquet i* pour éventuellement le renvoyer. Si on considère des délais de transmission de l'ordre de 500 ms (voire plus), un tel mécanisme est totalement inadapté au transfert de flux de données. On peut aussi remarquer qu'il sous-emploie la bande passante du réseau (figure .7). Depuis le début des années 2000 l'IETF met au point le protocole SCTP qui fournit des services similaires à ceux de TCP, en abandonne certains et apporte les nouvelles fonctionnalités adaptées aux nouveaux besoins.

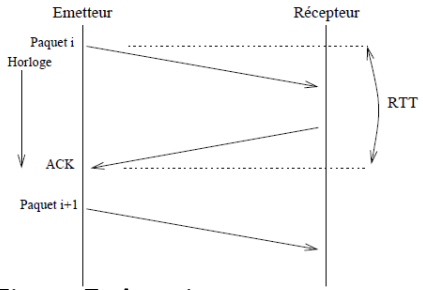


Figure.7 Acquittement par paquet

B. Acquittement par fenêtre de paquets

Le nombre de paquets à envoyer avant d'attendre le premier acquittement est fonction de deux paramètres : la largeur de la fenêtre précisée dans le champ WINDOW de l'en-tête. Des valeurs courantes sont de l'ordre de 4096, 8192 ou 16384 et la taille maximale des

données, ou MSS (Max. Segment Size) vaut 512 octets par défaut. C'est la plus grande taille du segment de données que TCP enverra au cours de la session (Fig.8) .

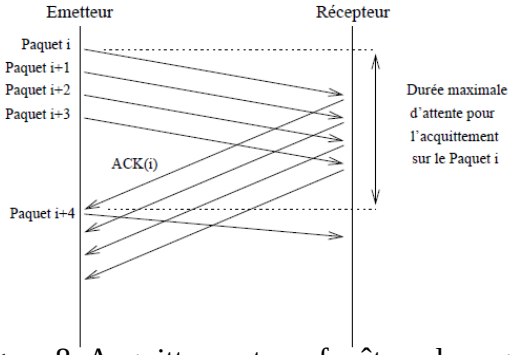


Figure.8 Acquittement par fenêtres de paquets

5 Le protocole UDP

UDP est l'acronyme de "User Datagram Protocol", il est défini dans la RFC 768 . Les données encapsulées dans un en-tête UDP sont des "paquets UDP".

Rappel : Au niveau de la couche *Internet* les datagrammes sont routés d'une machine à une autre en fonction des bits de l'adresse IP qui identifient le numéro de réseau. Lors de cette opération aucune distinction n'est faite entre les services ou les utilisateurs qui émettent ou reçoivent des datagrammes, tous les datagrammes sont mélangés.

La couche UDP ajoute un mécanisme qui permet l'identification du service (niveau *Application*). En effet, il est indispensable de faire un tri entre les diverses applications (services) : plusieurs programmes de plusieurs utilisateurs peuvent utiliser simultanément la même couche de transport et il ne doit pas y avoir de confusion entre eux. L'idée est d'associer la destination à la fonction qu'elle remplie. Cette identification se fait à l'aide d'un entier positif que l'on baptise **port**.

Pour communiquer avec un service distant il faut donc avoir connaissance de son numéro de port, en plus de l'adresse IP de la machine elle-même. On peut prévoir le numéro de port en fonction du service à atteindre. Un paquet UDP est conçu pour être encapsulé dans un datagramme IP et permettre un échange de données entre deux applications, sans échange préliminaire. Ainsi, si les données à transmettre n'obligent pas IP à fragmenter, un paquet UDP génère un datagramme IP et c'est tout !

– UDP apporte un mécanisme de gestion des ports, au dessus de la couche Internet.

– UDP est simplement une interface au dessus d'IP, ainsi l'émission des messages se fait-elle sans garantie de bon acheminement. Plus généralement, tous les défauts d'IP recensés sont applicables à UDP. Plus particulièrement, les paquets à destination d'une application UDP sont conservés dans une pile de type FIFO. Si l'application destinatrice ne les "consomme" pas assez rapidement, les plus anciens paquets risquent d'être écrasés par les plus récents.

- Il n'y a aucun retour d'information au niveau du protocole pour apporter un quelconque moyen de contrôle sur le bon acheminement des données. C'est au niveau applicatif qu'il convient de prendre en compte cette lacune.

– UDP est aussi désigné comme un mode de transport "non connecté", ou encore mode datagramme, par opposition à TCP ou SCTP, parmi les utilisations les plus courantes d'UDP on peut signaler le serveur de noms, base de données répartie au niveau mondial, et qui s'accommode très bien de ce mode de transport.

Chapitre 3 : Réseaux IP avancés

1 Le Tunnel IP

Le tunnel permet d'encapsuler un protocole dans un autre de même niveau ou supérieur. Précédemment, nous avons déjà analysé l'encapsulation des couches de la pile Arpa selon une progression naturelle de fonctionnalités. Ici, si le principe d'encapsulation est conservé, la logique initiale de construction, elle, est bousculée. Par exemple on peut envisager d'encapsuler IP dans de multiples protocoles autres qu'Ethernet (comme PPP, IP6,…) , dans une couche de transport comme TCP, voire même dans une couche applicative comme HTTP. Ce dernier exemple peut paraître " contre nature " et pourtant cela fonctionne.

Construire un tunnel a un coût : d'une part celui de la perte d'espace de données dans le datagramme (il faut loger un ou plusieurs en-têtes supplémentaires et le MTU reste constant lui !) et d'autre part celui du traitement supplémentaire (d´encapsulation, analyse) engendré par l'ajout de ces nouveaux en-têtes.

En résumé, construire un tunnel se traduit par une perte d'espace pour les données dans les datagrammes et par une consommation accrue de cycles cpus pour traiter les en-têtes supplémentaires. Heureusement le gain en fonctionnalités pour le réseau est substantiel.

Les tunnels qui transitent par une couche de transport sont gérés par une application (par exemple sshd ou httptunnel). Aussi le trafic de datagrammes remonte au niveau applicatif pour redescendre au niveau IP, ce qui a l'avantage de pouvoir être mis en œuvre par un utilisateur n'ayant pas nécessairement les droits de l'administrateur, mais par contre, outre une consommation supplémentaire de cycles cpu et des changements de contexte inhérents à l'architecture logicielle, a l'inconvénient d'être dédié à un seul port (fig.1).

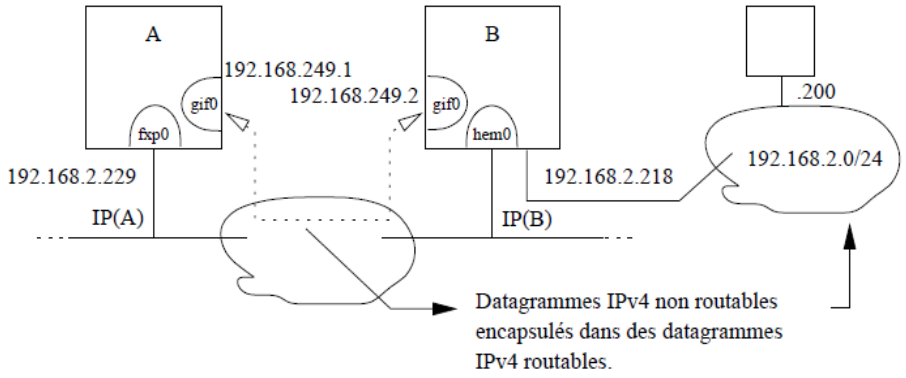


Figure. 1 Exemple de Tunnel générique (gif : generic tunnel interface)

La figure. 1 illustre l'encapsulation d'IP4 dans IP4 grâce à l'usage du " generic tunnel interface ". Il s'agit d'un pseudo-device (pas d'interface réel associé au device), qui permet d'encapsuler de l'IP (version 4 ou 6) dans de l'IP (version 6 ou 4). Le but de cet exemple de tunnel est de montrer un routage de datagrammes issus d'un réseau privé, le 192.168.2.0/24 , depuis la machine B (IP_B), vers la machine A (IP_A) et qui traverse un réseau public routé quelconque, non nommé sur la figure, de telle sorte que A soit intégrée au LAN 192.168.2.0/24.

- Par hypothèse la machine A sait comment router vers le 192.168.2.0/24. Un de ses interfaces réseaux peut être surchargé avec une adresse dans cette classe C.
- Le réseau 192.168.249.0/30 sert de réseau d'interconnexion entre les deux machines. Concrètement, il s'agit d'attribuer une adresse IP à chacun des pseudo-devices, qui ne soit pas déjà dans l'un des réseaux attachés à chacune des machines.

Conceptuellement, il serait parfaitement possible d'utiliser, par exemple, des adresses dans le 192.168.2.0/24, mais dans ce cas on préfère l'usage d'un réseau d'interconnexion qui permet de bien séparer fonctionnellement les adresses IP qui constituent le tunnel en lui-même de celles qui sont amenées à l'emprunter. De plus, si on souhaite (et c'est une quasi obligation quand on utilise des tunnels) ajouter un filtrage IP sur la machine B, il est beaucoup plus aisé pour la conception des règles de filtrage de considérer l'origine des datagrammes ayant une adresse source dans le 192.168.2.0/24 uniquement derrière le filtre.

Examinons maintenant quelle pourrait être la configuration spécifique à ce tunnel, Sur la machine A :

ifconfig gif0 create (Créer l'interface générique)
ifconfig gif0 inet tunnel IP(A) IP(B)
ifconfig gif0 inet 192.168.249.1 192.168.249.2 netmask 0xfffffffc
route add -net 192.168.2.0 192.168.249.2

Notez l'ajout de la route spécifique vers le réseau non directement raccordé. Puis, exécution des opérations symétriques sur la machine B :

ifconfig gif0 create
ifconfig gif0 inet tunnel IP(B) IP(A)
ifconfig gif0 inet 192.168.249.2 192.168.249.1 netmask 0xfffffffc

Notez que la 2ème ligne de configuration précise la source et la destination réelle des datagrammes alors que la 3ème indique l'adresse locale et distante des extrémités du tunnel. C'est une écriture particulière, adaptée au pilote de l'interface gif0 pour la configuration des tunnels.

Sur la machine B, on peut voir le résultat de la configuration comme ,ça :

$ifconfig gif0
gif0: flags=8011<UP,POINTOPOINT,MULTICAST> mtu 1280
tunnel inet IP(B) --> IP(A)
inet 192.168.249.2 -> 192.168.249.1 netmask 0xfffffffc
$ netstat -f inet -rn
...
192.168.249.1 192.168.249.2 UH 0 9779 - gif0

Et sur la machine A:

$ ifconfig gif0
gif0: flags=8011<UP,POINTOPOINT,MULTICAST> mtu 1280
tunnel inet IP(A) --> IP(B)
inet 192.168.249.1 -> 192.168.249.2 netmask 0xfffffffc
$ netstat -f inet -rn
...
192.168.2.0/24 192.168.249.2 UGS 0 83 gif0
192.168.249.2 192.168.249.1 UH 1 8941 gif0

Enfin, si on examine sur les interfaces hme0 puis gif0 de B le passage des datagrammes d'un ping, envoyés depuis A vers 192.168.2.200, l'observation pratique rejoint la théorie : on retrouve bien sur l'interface du tunnel (gif0) l'en-tête 2, décapsulé de son en-tête 1. Le datagramme est alors disponible au niveau de la pile IP de B pour être routé (routage direct ici) vers 192.168.2.200.

Le tableau qui suit résume le contenu des en-têtes observés :

Sur l'interface hme0

En-tête 1		En-tête 2
Src	IP_A	192.168.249.1
Dst	IP_B	192.168.2.200
Code	ipencap(4)	icmp

Sur l'interface gif0

En-tête	
Src	192.168.249.1
Dst	192.168.2.200
Code	icmp

2 Le Proxy

Le propos d'un proxy est de concentrer le trafic réseau via une seule machine pour toute une variété de protocoles (telnet, http, smtp, . . .). Il existe des proxy spécialisés sur tel ou tel protocole, qui effectuent donc des tâches potentiellement très complexes (par exemple squid pour http) ou très généraux et donc moins performants sur chaque protocole (nat au paragraphe suivant). Tout le trafic réseau qui passe par un proxy s'y arrête pour en repartir. Les conditions de ce " rebond " peuvent être paramétrées par des règles d'accès, ce qui en fait un élément utile en sécurité des réseaux. Le proxy joue aussi le rôle d'un médiateur, cache, optimiseur, … etc.

3 La Translation des adresses (NAT)

La pénurie d'adresses IP est à l'origine du besoin de translation des adresses. Son principe se trouve décrit dans la RFC 1631. Un tel dispositif se situe généralement à la frontière entre un réseau de type privé et un autre de type publique. Le cas le plus général est lorsque le réseau public est l'internet lui-même, et le réseau privé celui d'une entité quelconque abonnée aux services d'accès réseau d'un FAI, mais ce n'est pas une obligation conceptuelle.

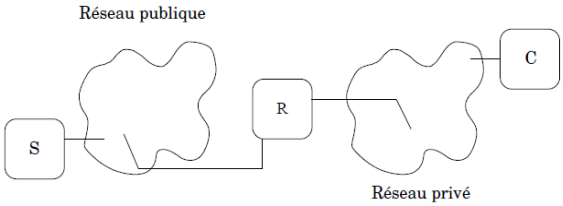


Figure. 2 Translation d'adresses

Sur la figure 2 le réseau privé comporte plus d'hôtes que d'adresses IP fournies dans le réseau public. Pour pouvoir se développer en s'affranchissant de cette contrainte, l'usage de la translation d'adresses et de ports (NAT et PAT) , ou encore NAPT comme " Network Address Port Translation " est incontournable parce que le réseau privé est bâti avec des adresses non routables potentiellement illimitées à l'échelle d'une entité privée, même grande.

R dispose de quelques adresses (un pool d'une adresse au minimum) routables sur le réseau public, qu'il peut assigner aux hôtes du réseau privé (C) qui initient une connexion vers le réseau publique (S). Cette assignation peut être dynamique ou statique. Un datagramme qui part de C vers S a une adresse source non exploitable sur le réseau public. R maintient une table, si C n'est pas déjà associé à une adresse routable du pool alloué à R,

celui-ci lui en attribue une et modifie à la volée l'adresse source du datagramme, de telle sorte que le retour de S puisse être routé convenablement jusqu'à R. Puis R modifie l'adresse de destination du datagramme pour lui donner l'adresse de C, sur le réseau privé. Si on fait l'hypothèse que la plupart des hôtes du réseau privé n'établissent pas simultanément des connexions vers le réseau public, le pool d'adresses publiques peut rester beaucoup plus petit que le nombre d'hôtes du réseau privé. Mais cette hypothèse est fragile considérant les besoins toujours plus grands d'accéder à l'information répartie.

Ce premier mécanisme se complète alors d'un second qui est le NAPT. En plus de traduire l'adresse IP à la volée, R attribue également un numéro de port différent. Ce dispositif autorise l'usage simultané d'une même adresse IP publique par des milliers d'hôtes du réseau privé. Le fonctionnement de la translation d'adresse et de port engendre une propriété intéressante pour R : il ne laisse passer aucun paquet du réseau public vers le réseau privé qui ne soit pas la réponse à une sollicitation venue du réseau privé, c'est donc en standard un fonctionnement à sens unique.

4 Filtrage IP

Le propos du filtrage IP est d'appliquer des règles de filtrage à un flux de datagrammes IP afin de prendre une décision qui est le plus souvent binaire : laisser passer ou ne pas laisser passer avec en option de conserver une trace de passage (des logs).

Par son usage on cherche à protéger un site ou une machine d'un flux de datagrammes pour lesquels on suspecte que tous ne sont pas envoyés par des utilisateurs bienveillants. Le filtre s'efforce d'éliminer le trafic indésirable à partir de considérations à priori, mais il ne représente pas la panacée en matière de sécurité sur les réseaux, autrement dit il ne faut pas penser qu'un filtre IP suffit à régler tous les problèmes de sécurité d'un site ou d'un hôte.
En effet, à partir du moment où le filtre laisse passer certains datagrammes, même à priori innocents, une porte est ouverte au détournement de l'usage initial du service offert. Dans ces conditions il faut se rendre à l'évidence : il n'y a pas de sécurité absolue sur le réseau !

Dans la littérature, un routeur filtrant est nommé " FireWall ", qu'il faut traduire en " pare-feux ". Le filtrage IP est une caractéristique essentielle de tout routeur digne de ce nom !
Il est aussi possible de faire du filtrage IP avec les Unix libres, c'est cette approche qui est choisie dans les paragraphes qui suivent parce qu'accessible à toutes les bourses. . .

Etablir des règles de filtrage IP sous-entend avoir une connaissance exhaustive de tous les éléments qui s'y rattachent :

- Noms des interfaces réseaux impliqués
- Protocoles réseaux employés (tcp, udp, icmp,. . .)
- Protocoles applicatifs autorisés (smtp, domain, http. . .)
- Adresses IP, numéro de ports, masques de sous-réseaux
- Sens du trafic par rapport aux interfaces ci-dessus

Il est assez aisé de mettre en place un filtrage simple, par contre cette opération peut devenir complexe dés lors qu'on utilise des protocoles applicatifs compliqués, mettant en jeux une stratégie d'utilisation des ports et des adresses non triviale.

Considérons un site simple, comme celui de la figure 3. La machine C accède depuis l'extérieur à la machine S, protégée par le filtrage IP activé sur la machine R, qui agit donc en tant que routeur filtrant.

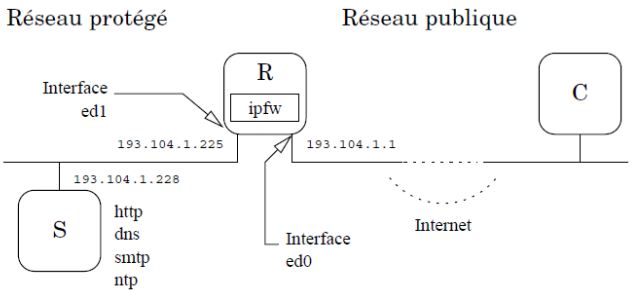


Figure. 3 Filtrage IP

Adaptons-y les règles du modèle de base, extraites du fichier /etc/rc.firewall de la configuration standard d'une machine unix (c'est un script shell). L'examen de ces règles nous permet de découvrir la nature du trafic autorisé ou non.

```
1 # −−− Interface externe
2 oif="ed0"
3 onet="193.104.1.0"
4 omask="255.255.255.224"
5 oip="193.104.1.1"
6
7 # −−− Interface interne
8 iif="ed1"
9 inet="193.104.1.224"
10 imask="255.255.255.224"
11 iip="193.104.1.225"
12
13 # −−− Ne pas laisser passer le ‘‘spoofing’’
14 ipfw add deny all from ${inet}:${imask} to any in via ${oif}
15 ipfw add deny all from ${onet}:${omask} to any in via ${iif}
16
17 # −−− Ne pas router les adresses de la RFC1918
18 ipfw add deny all from 192.168.0.0:255.255.0.0 to any via ${oif}
19 ipfw add deny all from any to 192.168.0.0:255.255.0.0 via ${oif}
20 ipfw add deny all from 172.16.0.0:255.240.0.0 to any via ${oif}
21 ipfw add deny all from any to 172.16.0.0:255.240.0.0 via ${oif}
22 ipfw add deny all from 10.0.0.0:255.0.0.0 to any via ${oif}
23 ipfw add deny all from any to 10.0.0.0:255.0.0.0 via ${oif}
24
25 # −−− Laisser passer les connexions TCP existantes
26 ipfw add pass tcp from any to any established
27
28 # −−− Permettre l'arrivée du courrier SMTP (25)
29 ipfw add pass tcp from any to 193.104.1.228 25 setup
30
31 # −−− Permettre l'accès au serveur HTTP (80)
32 ipfw add pass tcp from any to 193.104.1.228 80 setup
33
34 # −−− Rejetter et faire des logs de toutes les autres demandes de connexion
35 ipfw add deny log tcp from any to any in via ${oif} setup
36
37 # −−− Permettre l'établissement des autres connexions (via $iif).
38 ipfw add pass tcp from ${inet}:${imask} to any setup in via ${iif}
39
40 # −−− Permettre le trafic UDP/DOMAIN vers/depuis les serveurs DNS externes
41 ipfw add pass udp from any 53 to any 53
```

```
42
43 # −−− Permettre le trafic NTP vers/depuis les serveurs de dates
44 ipfw add pass udp from any 123 to any 123
45
46 # −−− Permettre le passage de tous les paquets ICMP
47 ipfw allow icmp from any to any
48
49 # −−− Tout ce qui n'est pas explicitement autorisé est
50 # implicitement interdit (cf comportement par défaut d'ipfw).
51 ipfw deny ip from any to any
```

Chapitre 4 : Protocoles Applicatifs

1. Le Serveur de noms (DNS)

S'il est obligatoire d'attribuer au moins une adresse IP à une machine pour pouvoir l'interconnecter au réseau avec d'autres machines de même nature, en revanche, lui attribuer un nom symbolique n'est absolument pas nécessaire au bon fonctionnement de trois couches basses ARPA. Ce nommage symbolique est simplement beaucoup plus naturel pour les utilisateurs que la manipulation des adresses IP, même sous forme décimale pointé .Il n'intervient donc qu'au niveau applicatif, ainsi la majeure partie des applications réseaux font usage de noms symboliques avec, de manière sous-jacente, une référence implicite à leur(s) correspondant(s) numérique(s). Le serveur de noms est en général le premier service mis en route sur un réseau, tout simplement parce que beaucoup de services le requièrent pour accepter de fonctionner (le courrier électronique en est un exemple majeur).

Au début de l'histoire de l'Internet, la correspondance entre le nom (les noms s'il y a des synonymes ou " alias ") et l'adresse d'une machine est placée dans le fichier /etc/hosts, présent sur toutes les machines (unix) . La forte croissance du nombre des machines, a rendu obsolète cette approche. L'espace de noms, préalablement non structuré, est désormais réorganisé de manière hiérarchique, sous forme d'un arbre (et non d'un graphe). Cette organisation entraine une hiérarchisation des noms de machines et des autorités qui ont le pouvoir de les nommer, de les maintenir. Chaque nœud de l'arbre porte un nom, la racine n'en a pas. Les machines, feuilles de l'arbre, sont nommées à l'aide du chemin parcouru de la feuille (machine) à la racine (non nommée). Le séparateur entre chaque embranchement, ou noeud, est le point décimal. Voici un exemple de nom de machine : www.sio.ecp.fr

a) Domaine et zone

Le réseau peut être considéré comme une hiérarchie de domaines. L'espace des noms y est organisé en tenant compte des limites administratives ou organisationnelles. Chaque nœud, appelé un domaine, est baptisé par une chaîne de caractères et le nom de ce domaine est la concaténation de toutes les étiquettes de nœuds lues depuis la racine, donc de droite à gauche. Par exemple :

fr →Domaine fr

ecp.fr→ Domaine ecp.fr sous domaine du fr

sio.ecp.fr→ Domaine sio.ecp.fr sous domaine de ecp.fr

Bien que le serveur de noms, " Domain Name Server " fasse référence explicitement au concept de domaine, pour bien comprendre la configuration d'un tel service il faut également comprendre la notion de " zone ". Une zone est un point de délégation dans l'arbre DNS, autrement dit une zone concerne un sous arbre du DNS dont l'administration lui est propre. Ce sous arbre peut comprendre plusieurs niveaux, c'est à dire plusieurs sous domaines. Une zone peut être confondue avec le domaine dans les cas les plus simples.

Dans les exemples ci-dessus, on peut parler de zone sio.ecp.fr puisque celle-ci est gérée de manière autonome par rapport à la zone ecp.fr.

Le serveur de noms est concerné par les " zones ". Ses fichiers de configuration précisent la portée de la zone et non du domaine. Chaque zone doit avoir un serveur principal (" master ") qui détient ses informations d'un fichier configuré manuellement ou semi manuellement (DNS dynamique). Plusieurs serveurs secondaires (" slave ") reçoivent une copie de la zone via le réseau et pour assurer la continuité du service (par la redondance des serveurs).

Le fait d'administrer une zone est le résultat d'une délégation de pouvoir de l'administrateur de la zone parente et se concrétise par la responsabilité de configurer et d'entretenir le champ SOA (" start of authority ") de la dite zone.

b) Hiérarchie des domaines

Cette organisation du nommage (figure .1) pallie aux inconvénients de la première méthode :

- Le NIC (Network Information Center) gère le plus haut niveau de la hiérarchie, appelé aussi celui des " top levels domains " (TLD).
- Les instances régionales du NIC gèrent les domaines qui leur sont dévolus. Par exemple le " NIC France " gère le contenu de la zone .fr.
- Chaque administrateur de domaine (universités, entreprises, associations, entités administratives,. . .) est en charge de son domaine et des sous domaines qu'il crée. Sa responsabilité est nominative vis-à-vis du NIC. On dit aussi qu'il a l'autorité sur son domaine (" authoritative for the domain ")

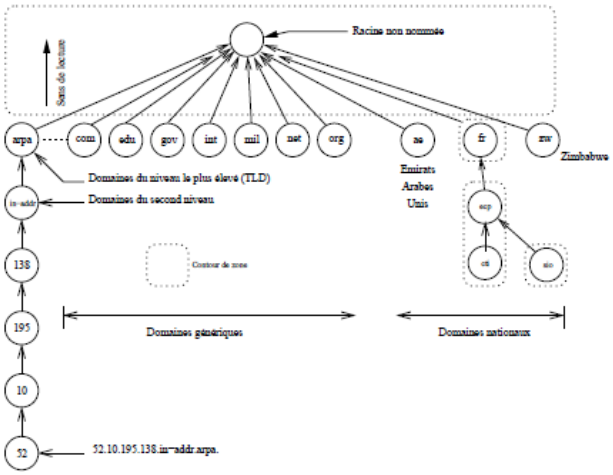


Figure.1 Organisation hiérarchique des domaines

- Les éventuels conflits de nommage sont à la charge des administrateurs de domaine. Du fait de la hiérarchisation, des machines de même nom peuvent se trouver dans des domaines différents sans que cela pose le moindre problème.

Chaque site raccordé de manière permanente procède de cette manière, ainsi il n'y a pas une base de données pour l'Internet mais un ensemble structuré de bases de données reliées entres elles et formant une gigantesque base de données distribuée.

c) Le resolver

Sur un même réseau logique on a coutume de ne pas utiliser le nom complet des machines auxquelles on s'adresse couramment et pourtant ¸ca fonctionne ! La raison est que le " resolver ", partie du système qui est en charge de résoudre les problèmes de conversion entre un nom de machine et son adresse IP, utilise un mécanisme de complétion (" domain completion ") pour compléter le nom de machine simplifié, jusqu'à trouver un nom plus complet que le serveur de noms saura reconnaître dans sa base de données. Le " resolver " connait par hypothèse le ou les noms de domaine (lus dans le fichier de configuration /etc/resolv.conf) qui concernent la machine locale. Une station de travail n'en a généralement qu'un seul alors qu'un serveur peut en comporter plusieurs, par exemple si on souhaite consolider toute une palette de services pour plusieurs domaines sur une même machine.

Exemple d'un tel fichier :
domain sio.ecp.fr
search sio.ecp.fr., ecp.fr.
nameserver 138.195.52.68
nameserver 138.195.52.69
nameserver 138.195.52.132
Plus généralement ce nom de domaine se présente sous forme *d*1.*d*2...*dn*. Ainsi, en présence d'un nom symbolique *x*, le " resolver " teste pour chaque i, *i* = {1, 2, . . . , *n*} l'existence de *x.di.di*+1...*dn* et s'arrête si celle-ci est reconnue.
Dans le cas contraire la machine en question n'est pas atteignable.
Exemple, avec le domaine ci-dessus :
a) machine = www (requête)
www.sio.ecp.fr → Succès !
b) machine = www.sio (requête)
www.sio.sio.ecp.fr → Echec !
www.sio.ecp.fr → Succès !

Le " resolver " désigne un ensemble de fonctions placées dans une bibliothèque standard qui font l'interface entre les applications et les serveurs de noms. Par construction les fonctions du " resolver " sont compilées avec l'application qui les utilise (physiquement dans la libc, donc accessibles par défaut).

Le " resolver " applique la stratégie locale de recherche, définie par l'administrateur de la machine, pour résoudre les requêtes de résolution de noms. Pour cela il s'appuie sur son fichier de configuration /etc/resolv.conf et sur la stratégie locale d'emploi des possibilités (serveur de noms, fichier /etc/nsswitch.conf, NIS,. . .).

d) Stratégie de fonctionnement

La figure. *2* illustre le fait que chaque serveur de noms a la maîtrise de ses données mais doit interroger ses voisins dés qu'une requête concerne une zone sur laquelle il n'a pas l'autorité de nommage. Ainsi, un hôte du domaine " R2 " qui veut résoudre une adresse du domaine " R1 " doit nécessairement passer par un serveur intermédiaire pour obtenir l'information. Cette démarche s'appuie sur plusieurs stratégies possibles,

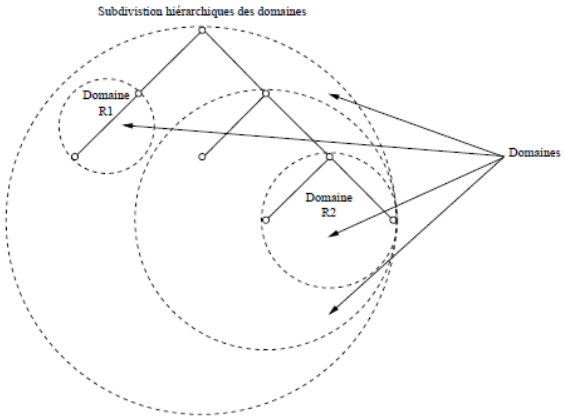


Figure. 2 Recherche de domaine

- Interrogation locale
  - Le processus demande l'adresse IP d'un serveur. Le " resolver " envoie la demande au serveur local.
  - Le serveur local reçoit la demande, parce qu'il a l'autorité sur le domaine demandé (le sien), il répond directement au " resolver ".

- Interrogation distante
  - Un processus demande l'adresse IP d'une machine. Le " resolver " envoie sa requête au serveur local.
  - Le serveur local reçoit la requête et dans ce deuxième cas il ne peut pas répondre directement car la machine n'est pas dans sa zone d'autorité, il interroge alors un serveur racine pour avoir l'adresse d'un serveur qui a l'autorité sur la zone demandée par le processus.
  - Le serveur racine renvoie l'adresse d'un serveur qui a officiellement l'autorité sur la zone

Un mécanisme de cache accélère le processus ci-dessus

- Interrogation par " procuration "

Le trafic destiné au serveur de noms peut consommer une partie non négligeable de la bande passante, c'est pourquoi il peut être stratégique de concentrer les demandes vers un seul serveur régional et donc de bénéficier au maximum de l'effet de cache décrit précédemment

2. Le Courrier électronique

Le courrier électronique, ou " mail " est l'un des deux services les plus populaires sur le réseau, avec le web. C'est aussi l'un des plus vieux services du réseau, bien avant que le réseau existe sous la forme que l'on pratique aujourd'hui. La préface de la [RFC 822], document fondamental parmi les documents fondamentaux pour ce chapitre, laisse supposer l'existence de nombreux formats d'´echanges électroniques sur l'Arpanet, et ce avant 1977. Sa popularité repose sur sa grande souplesse et rapidité d'emploi. Il permet aussi bien les échanges professionnels que les échanges privés ; son mode d'adressage donne la possibilité d'envoyer un courrier à une personne comme à une liste de personnes ou encore à un automate capable de rediffuser vers un groupe (" mailing-list ").

Tous les courriers électroniques ont un destinataire précisé par son adresse électronique, ou " E-mail ". Celle-ci précise le nom du destinataire et le site où il reçoit son courrier électronique. Le nom du destinataire est une chaîne de caractères. Le caractère " @ " (lire " at ") sépare l'identificateur du destinataire de la destination. La destination est peut être vide (il s'agit alors d'un destinataire sur la machine courante, ou d'un synonyme (" alias ") que le sendmail local sait traiter), peut être aussi un nom de machine du domaine local, le nom d'un autre domaine ou d'une machine sur un autre domaine.

Les adresses suivantes ont un format valide :

user1 : Destinataire local.

user2@nom_de_machine: Destinataire sur une machine du domaine courant (rappelez-vous, il existe un mécanisme de complétion dans le " resolver" !).

user3@nom_de_machine.domaine: Destinataire sur une machine particulière d'un domaine particulier (non forcément local).

a) Format d'un " E-mail "

Les octets qui composent un courrier électronique obéissent à une structure bien définie par la [RFC 822] : un en-tête et un corps de message, séparés par une ligne blanche (deux CRLF qui se suivent). Le contenu de l'en-tête dans son intégralité n'est pas toujours spontanément montré par les outils qui nous permettent de lire et d'envoyer du courrier électronique. Une option est toujours accessible pour ce faire.

Une partie de l'en-tête est générée automatiquement par le programme qui se charge du transfert (MTA) , une autre est ajoutée par le programme qui permet de composer le mail, le MUA, une autre enfin est tapée par l'utilisateur lui-même (Fig.3) .

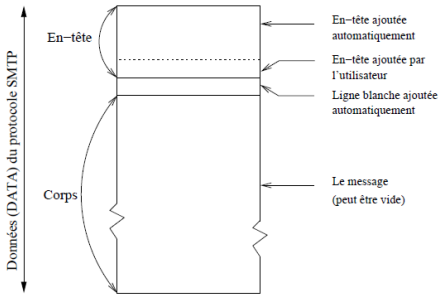


Figure. 3 Format d'un email

b) Quelques champs de l'en-tête

Type d'information	Noms des champs
Destinataire(s) du courrier	To, Cc, Bcc
Origine du courier	From, Reply-To
Identification du courrier	Message-ID
Cheminement du courrier	Received, Priority
Nature du contenu	Content-Transfer-Encoding, Content-Type
Divers	Date, Subject
Champs étendus	X-*

- To (The primary recipients) : Il s'agit du (des) destinataire(s) principaux du message (" recipient "). Ce champ peut être vide, le MTA prend alors une décision paramétrable pour le compléter.
- Cc (Carbon copy) : Ce champ est dans le fonctionnement un doublon de To, mais l'usage en nuance le sens : c'est une copie pour information qui est transmise au(x) destinataire(s) listé(s).
- Bcc (Blind carbon copy) : Une copie du message est transmise au(x) destinataire(s) listé(s), sans que les destinataires principaux soit informés.
- From (The sender) : il s'agit de l'émetteur du message. Le plus souvent il s'agit d'une seule personne, quand ce champ en liste plusieurs (le séparateur est la virgule " , ") le champ Sender doit préciser l'adresse de celui qui a effectivement envoyé le message.
- Reply-To (Alternative reply address) : Ce champ précise une adresse alternative à celle du champ From pour l'envoi de la réponse. Cette disposition est utilisée par les robots de gestion des mailing-list, pour distinguer l'auteur du message et le destinataire de la réponse.
- Message-ID (Unique identifier for message) : Ce champ est censé identifier de manière unique le message. Il est fabriqué dés sa soumission au premier MTA .

- Received (Trace routing of mail) : C'est une trace du routage suivi par le message, depuis sa soumission jusqu'à sa délivrance finale. Chaque MTA ajoute un champ de ce type. Le cheminement est à suivre en commençant en fin de l'en-tête.

- Priority : En fonction d'une valeur qui est urgent, normal ou non-urgent les messages qui ne peuvent être délivrés immédiatement sont placés dans une file d'attente dont la date d'expiration est d'autant plus courte que le message est urgent.
- Content-Transfer-Encoding: Indique comment est encodé le corps du message pour supporter les caractères hors jeu ascii 7 bits (SMTP ne transporte que des caractères 7 bits).

- Content-Type: Ce champ indique comment est constitué le corps du message. Par défaut il est supposé être constitue que de caractères 8 bits dont le bit de poids fort est sans signification (7 bits effectifs), Le cas contraire est celui d'un message qui contient des pièces jointes.

- Subject: C'est une courte chaîne de caractères qui résume le message.

c) Le Protocole SMTP

SMTP est un protocole ASCII (7 bits). La partie cliente de la transaction se connecte sur le port **25** du serveur et envoie des commandes auxquelles le serveur répond par des codes numériques qui indiquent le statut de la prise en compte de la commande. Expérimentalement on peut trouver quelques uns des mots réservés du protocole : HELO, MAIL, RCPT, DATA, QUIT, RSET et NOOP. Examinons succinctement l'usage de :

- Commande HELO

Syntaxe : HELO <espace> <domaine> <CRLF>
Cette commande est utilisée pour identifier l'émetteur du mail. L'argument qui suit, domain est le nom de la machine ou du domaine d'où provient la connexion. En réponse le serveur envoie une bannière dans laquelle il s'identifie et donne la date courante. Cette information est optionnelle, ce qui compte c'est le code de retour pour confirmer l'aptitude au travail du serveur !

- Commande MAIL

Syntaxe : MAIL <espace> FROM : <chemin inverse> <CRLF>
Cette commande débute un transfert de mail. En argument sont transmis (chemin inverse) l'adresse e-mail de l'émetteur et la liste des machines qui ont relayé le mail courant. La première machine est la plus récente. Cette information est utilisée pour renvoyer, s'il y a lieu, une notification de problème à l'émetteur du mail.
Par exemple : MAIL FROM:<@mailhub.ici:@mailhost.labas:Lambda@mondomain.fr>

- Etc…

d) Courriers indésirables (spam)

Le spam est l'aspect très désagréable du courrier électronique. Par " spam " on désigne ces innombrables courriers, le plus souvent à caractère commercial, qui envahissent nos boites aux lettres électroniques. Certaines estimations tablent sur au moins 30% de spam dans le trafic mail mondial et cette estimation est régulièrement revue à la hausse. Deux questions se posent, comment le caractériser et surtout comment l'éviter ?

- Un contenu commercial, publicitaire, financier, ou qui tente de retenir l'attention du lecteur à partir d'une histoire dont l'issue est toujours pécuniaire et au détriment du destinataire.
- Une importante liste de destinataires. Le champ Cc : peut contenir par exemple des centaines de destinataires.
- Un en-tête de message truque. Par exemple le champ Message-ID : qui est censé identifier le message de manière unique est absent ou incohérent

- Un grand nombre d'exemplaires du même message envoyé dans un court laps de temps.
- Utilisation de l'adresse d'un destinataire sans son consentement explicite pour ce type d'envoi.
- Le contenu du mail contient un virus, soit dans le corps du message soit dans une pièce jointe. Par abus ce genre de mail est parfois traité comme du spam.

3. Système de gestion de réseau

Un système de gestion de réseau (Network Management System) est une collection d'outils pour la surveillance et le contrôle afin de permettre à un opérateur d'effectuer la plupart des opérations de gestion depuis un interface le plus simple et ergonomique possible !
C'est un ensemble de logiciels (Network Management Entity) associés éventuellement à des matériels spécifiques, qui sont déployés sur tous les composants du système d'information. Un NMS est donc conçu pour donner une image unifiée du réseau, quelle que soit son étendue et son hétérogénéité. Le logiciel utilisé pour visualiser l'image du réseau est un NMA (Network Management Application, voir fig.4).
Un NME :

- Collecte des données sur l'activité réseau
- Conserve ces données dans une base
- Répond aux requêtes du NMA, notamment sur les points suivants :
  - Transmission des données collectées
  - Changement d'un paramètre de configuration
  - Fourniture de statut de composants logiciels ou matériels
  - Génération de tests
- Envoi des messages d'alerte (trap) en cas de détection d'événements exceptionnels.

Au moins un nœud du réseau est désigné comme étant le manager, et supportant le NMA. Cette architecture n'est pas nécessairement centralisée, la supervision du réseau peut s'effectuer par secteurs.

Le protocole SNMP (Simple Network Management Protocol)

SNMP est un terme un peu générique qui désigne à la fois un protocole réseau applicatif bien précis, une collection de spécifications pour le management de réseau et la définition de structures de données ainsi que leurs concepts associés. SNMP est né en 1988 de la nécessité de disposer d'un outil de supervision du réseau dés lors que celui-ci comporte un grand nombre d'hôtes qui interagissent, stations, serveurs, éléments de routage ou de commutation ou encore boites noires. Leur nombre grandissant sur les LANs (des machines en clusters par exemples) implique d'avoir un outil qui permette " d'expliquer " le réseau. Les logs, au sens de syslog , même concentrés, filtrés, et triés ne délivrent une information parfois trop verbeuse et en tout cas structurée différemment selon les applications ou les noyaux.

Enfin, n'importe quelle machine munie d'une stack IP est susceptible de supporter SNMP, depuis le calepin électronique, en passant par la borne wifi et jusqu'au mainframe.
En quelques mots, SNMP permet :

- De cartographier le réseau
- De fournir un panel très complet de résultats de mesures pour chaque hôte
- De mesurer en temps réel la consommation de ressources d'une application
- De signaler les dysfonctionnements
- De changer certains paramètres réseaux de fonctionnement

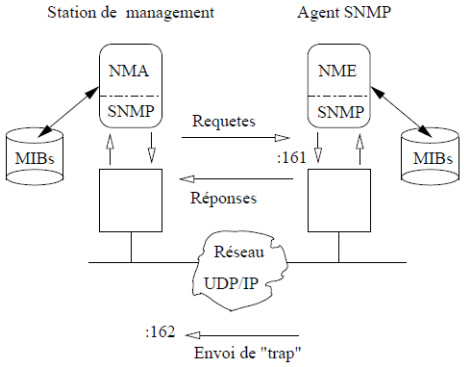


Figure. 4 Architecture SNMP

Un système d'exploitation peut être vu comme une vaste collection de compteurs et d'horloges auxquels SNMP nous permet d'accéder à distance pour les lire et les modifier (certaines, sous réserve d'y avoir accès). Afin que les agents et les managers soient interopérables les variables sont collectionnées selon une représentation arborescente très structurée et standardisée, ce sont les MIBs (" Management Information Base "). On les retrouve partout où SNMP est supporté. Ainsi, une même information se nomme de la même manière quelle que soit l'implémentation de SNMP et indépendamment de sa valeur qui est fonction du contexte. C'est donc très commode pour automatiser les traitements (scripts de collecte et de surveillance. . .) dans un réseau qui est hétérogène la plupart du temps !

Actuellement c'est la MIB-2 qui est la plus répandue (RFC 1213), elle répond parfaitement aux besoins élémentaires. Si un appareil ou un système a des besoins spécifiques il est toujours possible d'ajouter des branches au tronc commun, un embranchement est prévu pour cela, on parle alors d'une extension " vendor ".

La figure 4 présente la relation entre les deux entités logicielles qui dans le cas de SNMP se nomment :

- Agent SNMP, ou NME (le serveur) : C'est un logiciel qui s'exécute sur l'appareil que l'on souhaite administrer à distance. Il répond aux requêtes du gestionnaire, et génère des alarmes (traps) si besoin est. La configuration d'un agent est en général assez simple (par rapport à celle d'un logiciel Manager).

- Manager NMA (le client) C'est le logiciel qui s'exécute sur la station d'administration. Sa configuration est forcément plus délicate que celle de l'agent parce qu'il nécessite une adaptation au réseau local qui est toujours un cas particulier. Il existe de nombreux logiciels HP Open-View, SUN Net Manager , IBM Netview, Spectrum, ISM OpenMaster, SNMPc.. . .
- Sonde RMON (alternative de serveur) : Dans le cadre d'une architecture de supervision globale de réseau : si chaque agent sur chaque hôte peut répondre individuellement sur les événements réseau le concernant, il manque un maillon plus global qui fasse la supervision du réseau en lui-même, le véritable " networking management ". Cet élément existe, c'est ce qu'on appelle une sonde RMON, ou encore " Remote Monitoring ". C'est une entité logicielle, comme un agent SNMP. Elle s'appuie sur une extension de la MIB de base. On la trouve principalement sur les éléments de commutation ou de routage, l`a où se concentre le trafic réseau, mais on peut la trouver sur un hôte également, par exemple sur un serveur critique.

Chapitre 5 : Sockets et architectures de serveurs

1 Généralités

La version BSD 4.1c d'Unix pour VAX, en 1982, a été la première à inclure TCP/IP dans le noyau du système d'exploitation et à proposer une interface de programmation de ces protocoles : les sockets. Les sockets sont ce que l'on appelle une API (" Application Program Interface ") c'est à dire une interface entre les programmes d'applications et la couche transport, par exemple TCP ou UDP. Néanmoins les sockets ne sont pas liés à TCP/IP et peuvent utiliser d'autres protocoles comme AppleTalk, Xerox XNS, etc.

Les créateurs des sockets ont essayé au maximum de conserver la sémantique des primitives systèmes d'entrées/sorties sur fichiers comme open, read, write, et close. Cependant, les mécanismes propres aux opérations sur les réseaux les ont conduits à développer des primitives complémentaires (par exemple les notions de connexion et d'adresse IP n'existent pas lorsque l'on a besoin d'ouvrir un fichier !). Comme pour un fichier, chaque socket actif est identifié par un petit entier appelé descripteur de socket. Unix place ce descripteur dans la même table que les descripteurs de fichiers, ainsi une application ne peut-elle pas avoir un descripteur de fichier et un descripteur de socket de même valeur.

Pour créer une socket, une application utilisera la primitive socket et non open, pour les raisons que nous allons examiner. En effet, il serait très agréable si cette interface avec le réseau conservait la sémantique des primitives de gestion du système de fichiers sous Unix, malheureusement les entrées/sorties sur réseau mettent en jeux plus de mécanismes que les entrées/sorties sur un système de fichiers, ce n'est donc pas possible. Il faut considérer les points suivants :

1. Dans une relation du type client-serveur les fonctions ne sont pas symétriques. Démarrer une telle relation suppose que le programme sait quel rôle il doit jouer.
2. Une connexion réseau peut être du type connectée ou non. Dans le premier cas, une fois la connexion établie le processus origine discute uniquement avec le processus destinataire. Dans le cas d'un mode non connecté, un même processus peut envoyer plusieurs datagrammes à plusieurs autres processus sur des machines différentes.
3. Une connexion est définie par un quintuplé qui est beaucoup plus compliqué qu'un simple nom de fichier.
4. L'interface réseau supporte de multiples protocoles comme XNS, IPX, APPLETALK, la liste n'est pas exhaustive. Un sous système de gestion de fichiers sous Unix ne supporte qu'un seul format.

En conclusion de ce paragraphe on peut dire que le terme *socket* désigne, d'une part un ensemble de primitives, on parle des *sockets de Berkeley*, et d'autre part l'extrémité d'un canal de communication (point de communication) par lequel un processus peut émettre ou recevoir des données. Ce point de communication est représenté par une variable entière, similaire à un descripteur de fichier.

2. Etude des primitives

Ce paragraphe est consacré à une présentation des primitives essentielles pour programmer des applications en réseaux.

a) Création d'un socket

La création d'un socket se fait par l'appel système socket.

```
#include <sys/types.h> /* Pour toutes les primitives de ce chap. il faut inclure ces fichiers. */
#include <sys/socket.h>
```

#include <netinet/in.h>
int socket(int PF, int TYPE, int PROTOCOL) ;
PF : Spécifie la famille de protocole (“ Protocol Family ”) à utiliser avec le socket. On trouve (extrait) par exemple sur FreeBSD 4 7.0 :

PF INET : Pour les sockets IPv4
PF INET6 : Pour les sockets IPv6
PF SNA : Pour le protocole SNA d’IBM
PF IPX : Protocole Internet de Novell
PF ATM : “ Asynchronous Transfert Mode ”

. . . . . .
TYPE : Cet argument spécifie le type de communication désiré. En fait avec la famille PF INET, le type permet de faire le choix entre un mode connecté, un mode non connecté ou une intervention directe dans la couche IP :

SOCK STREAM : Mode connecté Couche transport
SOCK DGRAM : Mode non connecté Idem
SOCK RAW : Dialogue direct avec la couche IP

PROTOCOL : Ce troisième argument permet de spécifier le protocole à utiliser. Il est du type UDP ou TCP le plus couramment.

IPPROTO TCP : TCP
IPPROTO SCTP : SCTP
IPPROTO UDP : UDP
IPPROTO RAW, IPPROTO ICMP : uniquement avec SOCK RAW

La primitive socket retourne un entier qui est le descripteur du socket nouvellement créé par cet appel. Si la primitive renvoie -1, la variable globale errno donne l’indice du message d’erreur. Enfin, quand un processus a fini d’utiliser une socket il appelle la primitive close avec en argument le descripteur du socket :
close(*descripteur du socket*) ;

b) La primitive bind

La primitive bind permet d’associer un socket à un couple (adresse IP, numéro de port) associés dans une structure de type sockaddr_in, pour IPv4. Mais la primitive bind est généraliste, ce qui explique que son prototype fasse état d’une structure générique nommée.
int bind(int sockfd, struct sockaddr *myaddr, socklen_t addrlen) .
sockfd : Usage du descripteur renvoyé par socket.
myaddr : La structure qui spécifie l’adresse locale que l’on veut associer au socket préalablement ouvert.
addrlen : Taille en octets de la structure qui contient l’adresse.
struct sockaddr { /* La structure */
uint8_t sa_len ; /* generique */
sa_family_t sa_family ;
char sa_data[14] ;
} ;

Bind retourne 0 si tout va bien, -1 si une erreur est intervenue. Dans ce cas la variable globale errno est positionnée à la bonne valeur.

a. La primitive connect

La primitive connect permet d’établir la connexion avec un socket distant, supposé à l’écoute sur un port connu à l’avance de la partie cliente. Son usage principal est d’utiliser un mode “ connecté ”. L’usage d’un socket en mode datagramme est possible mais à un autre sens (voir plus loin) et est moins utilisé. La primitive connect a le prototype suivant :

int connect(int sockfd,const struct sockaddr *servaddr,socklen t addrlen) ;
sockfd : Le descripteur de socket renvoyé par la primitive socket.
servaddr : La structure qui définit l'adresse du destinataire, du même type que pour bind.
addrlen : La longueur de l'adresse, en octets.

c) Envoyer des données

Une fois qu'un programme d'application a créé un socket, il peut l'utiliser pour transmettre des données. Il y a cinq primitives possibles pour ce faire : send, write, writev, sendto, sendmsg

- *Envoi en mode connecté*

Send, write et writev fonctionnent uniquement en mode connecté, parce qu'elles n'offrent pas la possibilité de préciser l'adresse du destinataire. Les différences entre ces trois primitives sont mineures.

ssize_t write(int descripteur, const void *buffer, size_t longueur) ;
Quand on utilise write, le descripteur désigne l'entier renvoyé par la primitive socket. Le buffer contient les octets à transmettre, et leur cardinal.
La primitive send à la forme suivante :
int send(int s, const void *msg, size_t len, int flags) ;
s Désigne l'entier renvoyé par la primitive socket.
msg Donne l'adresse du début de la suite d'octets à transmettre.
len Spécifie le nombre d'octets à transmettre.
flags : drapeaux permettant de paramétrer la transmission du datagramme, notamment si le buffer d'écriture est plein ou si l'on désire, par exemple et avec TCP, faire un envoi en urgence (out-of-band) :

- *Envoi en mode datagramme*

Les deux autres primitives, sendto et sendmsg donnent la possibilité d'envoyer un message via une socket en mode non connecté. Toutes deux réclament que l'on spécifie le destinataire à chaque appel.

Ssize_t sendto(int s,const void *msg,size t len,int flags, const struct sockaddr *to, socklen_t tolen) ;
Les quatre premiers arguments sont exactement les mêmes que pour send, les deux derniers permettent de spécifier une adresse et sa longueur avec une structure du type sockaddr, comme vu précédemment avec bind.
Le programmeur soucieux d'avoir un code plus lisible pourra utiliser la deuxième primitive :
ssize_t sendmsg(int sockfd, const struct msghdr *messagestruct,int flags) ;
Où messagestruct désigne une structure contenant le message à envoyer, sa longueur, l'adresse du destinataire et sa longueur. Cette primitive est très commode à employer avec son pendant recvmsg car elle travaille avec la même structure.

d) Recevoir des données

Symétriquement aux cinq primitives d'envoi, il existe cinq primitives de réception : read, readv, recv, recvfrom, recvmsg.

- *Mode connecté*

ssize_t read(int descripteur, void *buffer,size_t longueur) ;
Buffer et longueur spécifie respectivement le buffer de lecture et la longueur de ce que l'on accepte de lire. Le programmeur peut aussi employer le readv, avec la forme :
Ssize_t readv(int descripteur, const struct iovec *iov, int vectorlen) ;
En addition aux deux primitives conventionnelles, il y a trois primitives nouvelles pour lire des messages sur le réseau :

ssize_t recv(int s, void *buf, size_t len, int flags) ;
s : L'entier qui désigne la socket.
buf : Une adresse où l'on peut écrire, en mémoire.
len : La longueur du buffer.
flags : Permet au lecteur d'effectuer un contrôle sur les paquets lus.

- *Mode datagramme*

ssize_t recvfrom(int s, void *buf,size_t len, int flags,struct sockaddr *from, socklen_t *fromlen) ;
Les deux arguments additionnels par rapport à recv sont des pointeurs vers une structure de type sockaddr et sa longueur. Le premier contient l'adresse de l'émetteur. Notons que la primitive sendto fournit une adresse dans le même format, ce qui facilite les réponses.
La dernière primitive recvmsg est faite pour fonctionner avec son homologue sendmsg :
ssize_t recvmsg(int sockfd, struct msghdr *messagestruct,int flags) ;

e) Autres primitives

Imaginons un serveur en train de répondre à un client, si une requête arrive d'un autre client, le serveur étant occupé, la requête n'est pas prise en compte, et le système refuse la connexion. La primitive listen est là pour permettre la mise en file d'attente des demandes de connexions. Elle est généralement utilisée après les appels de socket et de bind et immédiatement avant le accept.
int listen(int sockfd, int backlog) ;
sockfd : l'entier qui décrit la socket.
backlog : Le nombre de connexions possibles en attente (quelques dizaines).

Comme déjà vu, un serveur utilise les primitives socket, bind et listen pour se préparer à recevoir les connexions. Il manque cependant à ce trio le moyen de dire au protocole " j'accepte désormais les connexions entrantes ". La primitive accept est le chaînon manquant !
int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen) ;

Dans le cas du mode connecté on termine une connexion avec la primitive close ou shutdown. **int close(descripteur) ;** mais la primitive shutdown permet un plus grand contrôle sur les connexions en " full-duplex ". **int shutdown(int sockfd, int how) ;**
how permet de fermer partiellement le descripteur suivant les valeurs qu'il prend

3. Schéma général d'une session client – serveur

Pour illustrer l'enchainement des primitives relatives à l'utilisation des sockets Il est important de donner un aperçu de la structure d'un serveur et d'un client, mettant en œuvre les APIs vus dans ce chapitre, et de rapprocher les évènements réseaux de ceux observables sur le système et dans le processus qui s'exécute.

A. Relation client–serveur en mode connecté

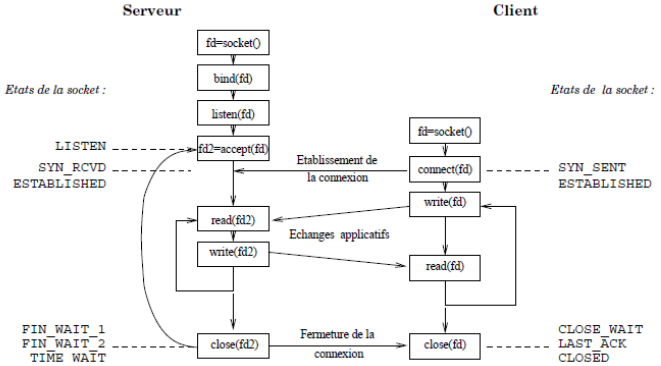


Figure. 1 Relation Client – serveur en mode connecté

La RFC 793 précise 11 états pour un socket et la figure .1 les met en situation de manière simplifiée. Ces états peuvent être visualisés avec la commande netstat -f inet [-a], dans la colonne state de la sortie. Les états sont :
LISTEN : Le socket du serveur est en attente passive d'une demande de connexion (ouverture passive).
SYN-SENT : C'est l'état du socket client qui a envoyé le premier paquet de demande d'une connexion avec un flag SYN mais non encore acquitté (ouverture active).
SYN-RCVD : Le socket du serveur à reçu un paquet de demande de connexion, l'acquitte et envoi son propre demande de connexion. Elle attend l'acquittement de sa demande.
ESTABLISHED : Les demandes de connexion sont acquittées aux deux extrémités.
FIN-WAIT-1 : Celui qui est à l'initiative de l'envoi du premier paquet de demande de fin est dans cet état (fermeture active).
FIN-WAIT-2 : On a reçu l'acquittement à la demande de fin de connexion.
TIME-WAIT : Le socket était en FIN-WAIT-2 et a reçu la demande de fin du socket distant. On doit attendre le temps suffisant pour être certain que le socket distant a bien reçu l'acquittement (réémission sinon).
CLOSE-WAIT : Le socket était en ESTABLISHED et a reçu une demande de fin. Cet état perdure jusqu''a ce que le socket envoie à son tour une demande de fin (fermeture passive).
LAST-ACK : La dernière demande de fin est en attente du dernier acquittement.
CLOSED : Etat de fin.

B. Relation client–serveur en mode non connecté

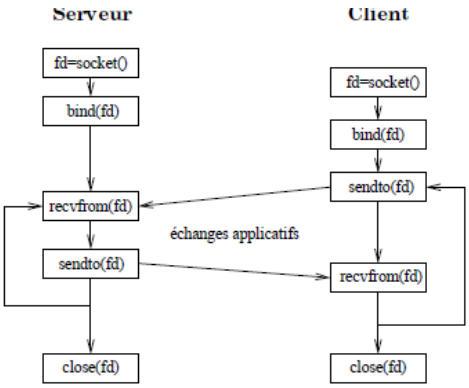


Figure. 2 Relation Client – serveur en mode non connecté

BIBLIOGRAPHIE /MEDIATHEQUE

REFERENCES

- Guy Pujolle, Cours réseaux et télécoms avec exercices corrigés, Eyrolles, 2008,
- François Laissus, Cours d'introduction à TCP/IP, 2009
- Stéphane Lohier, Dominique Présent, Transmissions et réseaux, Dunod 2010
- Andrew Tanenbaum, Réseaux Cours et exercices Dunod, 1999
- Douglas Comer, TCP/IP : Architecture, protocoles et applications, dunod 2001

WEBOGRAPHIE

- ✓ http://www.siteduzero.com/informatique/tutoriels/les-sockets/manipulation-de-sockets

- ✓ http://www.cisco.com/cisco/web/support/CA/fr/TsMrd.html

- ✓ Glossaire des protocoles: http://www.lincoste.com/ebooks/pdf/informatique/Glossaire_protocoles_reseau.pdf
- ✓ Exercices IP4 : http://www.inetdoc.net/pdf/adressage.ipv4.pdf
- ✓ Le protocole IP6 : http://www.commentcamarche.net/contents/524-le-protocole-ipv6

REPUBLIQUE TUNISIENNE
*****
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE
*****
DIRECTION GENERALE DES ETUDES TECHNOLOGIQUES
*****
INSTITUT SUPERIEUR DES ETUDES TECHNOLOGIQUES DE CHARGUIA
*****
Département Technologies de l'Informatique

TRAVAUX DIRIGES

ARCHITECTURES ET PROTOCOLES RESEAUX

Réalisé par :

MOHSEN BEN HASSINE

Année Universitaire 2011 - 2012

TD 1 : LES PROTOCOLES IP ,TCP ET UDP

Exercice N°1 :

- Pour chaque adresse IP suivante, préciser la classe et le masque associé

Adresses	Masque standard	Classe A	Classe B	Classe C	ClasseD OU E
12.144.44.15		☐	☐	☐	☐
192.168.0.1		☐	☐	☐	☐
130.45.11.57		☐	☐	☐	☐
83.206.12.34		☐	☐	☐	☐
240.100.90.50		☐	☐	☐	☐
221.122.66.5		☐	☐	☐	☐
11.11.116.15		☐	☐	☐	☐
111.111.116.115		☐	☐	☐	☐
129.1. 6.15		☐	☐	☐	☐
229.441. 6.15		☐	☐	☐	☐

- Donner aussi les adresses réseaux, les adresses de broadcast et le nombre de machines possibles (pour des masques standards)

Exercice N°2 :

On veut diviser le réseau principal 171.11.0.0 / 16 en 6 sous-réseaux distincts. Indiquez, pour chaque sous réseau :

- Le masque sous réseau
- La première et la dernière adresse
- L'adresse de broadcast
- et L'adresse de chaque sous-réseau

Exercice N°3 :

SNOOP est un packet sniffer en ligne de commande qui permet d'obtenir le détail du trafic visible depuis une interface réseau d'une machine. Dans la suite on présente la trace d'une communication point à point prélevée par ce logiciel :

ETHER: ----- Ether Header -----
ETHER: Packet 3 arrived at 11:42:27.64
ETHER: Packet size = 64 bytes
ETHER: Destination = 8:0:20:18:ba:40, Sun
ETHER: Source = aa:0:4:0:1f:c8, DEC (DECNET)
ETHER: Ethertype = 0800 (IP)
IP: ----- IP Header -----
IP: Version = 4
IP: Header length = 20 bytes
IP: Type of service = 0x00
IP: x xx. .... = 0 (precedence)
IP: ...0 .... = normal delay
IP: .... 0... = normal throughput
IP: .... .0.. = normal reliability
IP: Total length = 40 bytes
IP: Identification = 41980
IP: Flags = 0x4

IP: .1.. .... = do not fragment
IP: ..0. .... = last fragment
IP: Fragment offset = 0 bytes
IP: Time to live = 63 seconds/hops
IP: Protocol = 6 (TCP)
IP: Header checksum = af63
IP: Source address = 163.173.32.65, papillon.cnam.fr
IP: Destination address = 163.173.128.212, jordan
IP: No options
TCP: ----- TCP Header -----
TCP: Source port = 1368
TCP: Destination port = 23 (TELNET)
TCP: Sequence number = 143515262
TCP: Acknowledgement number = 3128387273
TCP: Data offset = 20 bytes
TCP: Flags = 0x10
TCP: ..0. .... = No urgent pointer
TCP: ...1 .... = Acknowledgement
TCP: .... 0... = No push
TCP: .... .0.. = No reset
TCP: .... ..0. = No Syn
TCP: ....0 = No Fin
TCP: Window = 32120
TCP: Checksum = 0x3c30
TCP: Urgent pointer = 0
TCP: No options
TELNET: ----- TELNET: -----
TELNET: ""

a) Quels sont les protocoles utilisés pendant cette communication, associer chaque protocole à sa couche ARPA
b) Indiquer les adresses MAC et IP utilisées
c) Indiquer les tailles des entêtes et des données pour chaque protocole
d) Expliquer : Time to live = 63 seconds/hops, Sequence number = 143515262, Acknowledgement number = 3128387273, TCP: Window = 32120, Checksum = 0x3c30 TCP: No options

CORRECTION TD 1

Exercice N°1 :

Adresses	Masque standard	Classe A	Classe B	Classe C	ClasseD OU E
12.144.44.15	/8	☒	☐	☐	☐
192.168.0.1	/24	☐	☐	☒	☐
130.45.11.57	/16	☐	☒	☐	☐
83.206.12.34	/8	☒	☐	☐	☐
240.100.90.50	/24	☐	☐	☐	☒
221.122.66.5	/24	☐	☐	☒	☐
11.11.116.15	/8	☒	☐	☐	☐
111.111.116.115	/8	☒	☐	☐	☐
129.1. 6.15	/16	☐	☒	☐	☐
~~229.441. 6.15~~	/24	☐	☐	☐	☐

Adresse Machine	Adresse réseau	Adresse broadcast	Nombre de machines
12.144.44.15	12.0.0.0	12.255.255.255	$2^{\wedge 24}$
192.168.0.1	192.168.0.0	192.168.0.255	256
130.45.11.57	130.45.0.0	130.45.255.255	$2^{\wedge 16}$
83.206.12.34	83. 0.0.0	83. 255.255.255	$2^{\wedge 24}$
240.100.90.50	240.100.90.0	240.100.90.255	256
221.122.66.5	221.122.66.0	221.122.66.255	256
11.11.116.15	11.0.0.0	11.255.255.255	$2^{\wedge 24}$
111.111.116.115	111.0.0.0	111.255.255.255	$2^{\wedge 24}$
129.1.6.15	129.1.0.0	129.1.255.255	$2^{\wedge 16}$
~~229.441.6.15~~	ERR	ERR	ERR

Exercice N°2 :

Pour diviser le réseau principal 171.11.0.0 / 16 en 6 sous-réseaux distincts, il faut utiliser le masque sous réseau /19 çad 255.255.224.0

N° sous-réseau	Adresse SR_i	1ère Adresse	Dernière adresse	Adresse de broadcast
1	172.16.0.0/19	172.16.0.1	172.16.0.254	172.16.0.255
2	172.16.32.0/19	172.16.32.1	172.16.32. 254	172.16.32. 255
3	172.16.64.0/19	172.16.64.1	172.16.64. 254	172.16.64. 255
4	172.16.96.0/19	172.16.96.1	172.16.96. 254	172.16.96. 255
5	172.16.128.0/19	172.16.128.1	172.16.128. 254	172.16.128. 255
6	172.16.160.0/19	172.16.160.1	172.16.160. 254	172.16.160. 255

Exercice N°3 :

a) Les protocoles utilisés pendant cette communication sont :

ETHERNET ➔ couche Interface
IP ➔ couche internet
TCP ➔ couche Transport
TELNET ➔ couche Application

b) Les adresses MAC et IP utilisées
   Adresses MAC :
   ETHER: Destination = 8:0:20:18:ba:40, Sun
   ETHER: Source = aa:0:4:0:1f:c8, DEC (DECNET)
   Adresses IP :
   IP: Source address = 163.173.32.65, papillon.cnam.fr
   IP: Destination address = 163.173.128.212, jordan

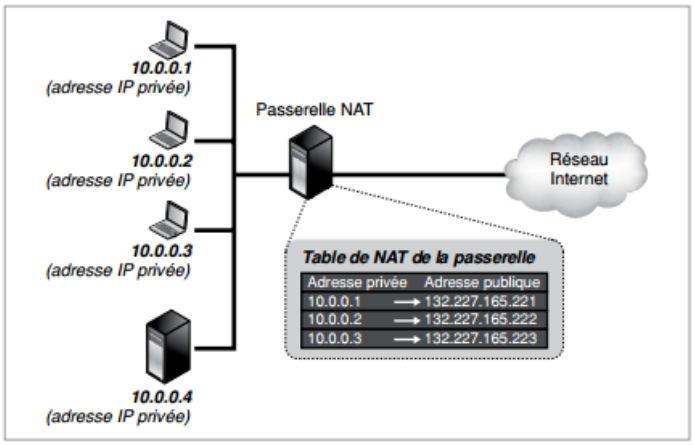
c) Les tailles des entêtes et des données pour les protocoles Ethernet et IP

   ETHER: Packet size = 64 bytes
   IP: Header length = 20 bytes
   IP: Total length = 40 bytes

d) Time to live = 63 seconds/hops ➔ temps de survie du paquet pour chaque routeur, Sequence number = 143515262 et Acknowledgement number = 3128387273 sont respectivement les n° des paquets émis et acquittés
   TCP: Window = 32120 ➔ longueur de la fenêtre d'acquittement
   Checksum = 0x3c30 ➔ code de vérification des entêtes
   TCP: No options ➔ pas d'options dans la paquet envoyé

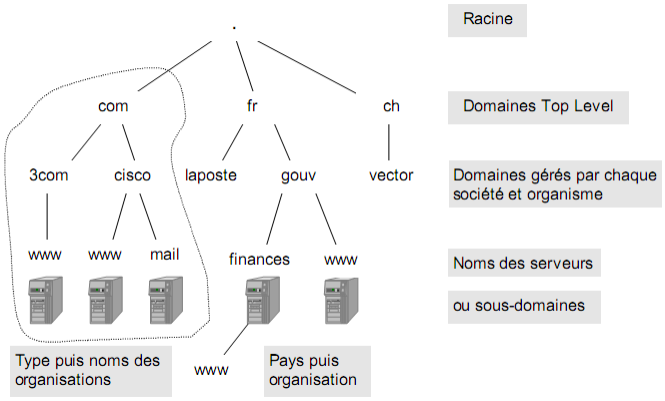
TD 2 : Protocoles Applicatifs

Exercice 1 :



- Que présente le schéma ci-dessus ?
- A quoi sert une passerelle NAT ?, que veut dire adresse privée et adresse publique ?

Exercice 2 :



- Enumérer tous les domaines possibles
- Quel protocole désigne les termes : WWW, mail
- Que veut dire SOA ? , zone ?, domaine ? , donner pour le cas de figure ci-dessus des exemples

Exercice 3 :

Indiquer le rôle des instructions suivantes :

- #include <sys/socket.h>
- #define PORTS 2058
- sock=socket(AF_INET,SOCK_STREAM,IPPROTO_TCP);
- bind(sock, (struct sockaddr *)&adr_s, sizeof(adr_s));
- listen(sock,5);
- nsock = accept(sock, (struct sockaddr *)&adr_c, &lg);
- read(nsock,buf,20);
- write(nsock,buf,20);
- close (nsock);

CORRECTION TD 2

Exercice N°1 :

- Le schéma présente une passerelle NAT qui permet d'établir une translation d'adresses privées de 4 postes en une adresse publique
- Une passerelle NAT est un dispositif qui se situe généralement à la frontière entre un réseau de type privé et un autre de type publique. Le cas le plus général est lorsque le réseau public est l'internet lui-même, et le réseau privé celui d'une entité quelconque abonnée aux services d'accès réseau d'un FAI.
- Une adresse publique est reconnue partout sur internet, par contre une adresse privée ne peut servir qu'à l'intérieur d'un réseau privé.

Exercice 2 :

- www.3com.com, www.cisco.com, mail.cisco.com, www.finances.gouv.fr, www.gouv.fr, vector.ch
- www➔ http, mail➔ smtp,pop,…
- Le SOA (" start of authority ") : Le pouvoir de l'administration de la zone parente.
- Zone : domaine où une organisation détient le pouvoir pour attribuer d'autres sous-domaines, exemple : finances.gouv.fr
- Domaine : Le réseau peut être considéré comme une hiérarchie de domaines. L'espace des noms y est organisé en tenant compte des limites administratives ou organisationnelles. Chaque nœud, appelé un domaine, est baptisé par une chaîne de caractères et le nom de ce domaine est la concaténation de toutes les étiquettes de nœuds lues depuis la racine, donc de droite à gauche. Par exemple : gouv.fr

Exercice 3 :

- #include <sys/socket.h>
  Intégration de la bibliothèque de gestion des sockets
- #define PORTS 2058
  La variable PORTS désigne le port utilisé pour ouvrir une voie de communication sous forme d'un entier (2058)
- sock=socket(AF_INET,SOCK_STREAM,IPPROTO_TCP);
  Création d'un socket IP4 en mode connecté utilisant TCP au niveau couche transport
- bind(sock, (struct sockaddr *)&adr_s, sizeof(adr_s));
  Définition de l'adresse IP et du port de connexion dans la structure sockaddr
- listen(sock,5);
  Ecoute des connexions par le serveur (5 connections au max. sont autorisés)
- nsock = accept(sock, (struct sockaddr *)&adr_c, &lg);
- Acceptation d'une demande de connexion
- read(nsock,buf,20);
  Réception des données dans une zone buffer de 20 octets

- write(nsock,buf,20);
  Envoi de la zone buffer (20 octets au max.)
- close (nsock);
  Fermeture du socket (libération de la mémoire)