Compléments - 1 : quelques commandes (Unix) pour observer l'Internet

Service de noms DNS : `nslookup`

Passage du nom de domaine (ex : thales.ujf-grenoble.fr) à l'adresse IP et vice versa

nslookup <nom de domaine>
nslookup <adresse IP>

Le fonctionnement de ce service sera expliqué dans le cours suivant

Test, temps de propagation : `ping`

ping <nom de domaine>	indique si l'hôte est actif (réponse avant délai de garde)
ping -s <nom de domaine>	donne aussi le temps moyen de transmission et les statistiques sur les pertes de paquets

Détermination du chemin suivi dans le réseau et des temps intermédiaires : `traceroute`

traceroute <nom de domaine> voir `man` pour les détails

Lecture des tables gérées par le protocole ARP (correspondance adresse IP - adresse Ethernet) : `arp`

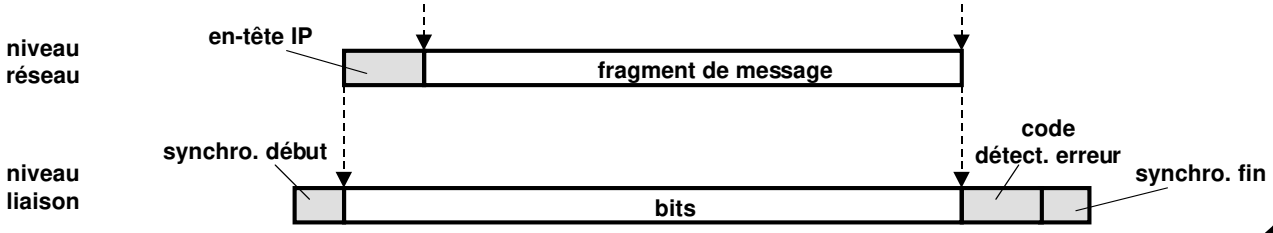
arp <nom de domaine>	pour un hôte particulier
arp -a	tous les hôtes de l'Ethernet

Consulter le fichier /etc/hosts

liste des hôtes administrés localement (y compris imprimantes, terminaux X, routeurs)

Compléments - 2 : retour sur les protocoles en couches

- **Comment les protocoles en couches traitent les messages**
  - Chaque couche rajoute au message transmis les informations nécessaires au fonctionnement du protocole correspondant
  - Ces informations sont analogues à une "enveloppe" placée par le protocole de niveau *i* pour encapsuler le message transmis par le niveau *i* + 1.
  - Le message est "mis sous enveloppe" à l'émission, "extrait de l'enveloppe" à la réception
- **Exemples**
  - Niveau réseau
    - Les messages (définis au niveau transport) sont découpés et encapsulés dans des paquets. Enveloppe = informations de routage (adresses), informations de fragmentation
  - Niveau liaison
    - Les suites de bits qui constituent les paquets sont encapsulées dans des trames. Enveloppe = informations de délimitation (début et fin), informations de redondance (contrôle d'erreur)



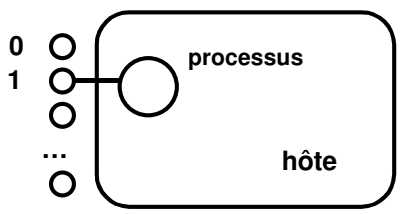
Protocoles de transport

- **Fonctions**
  - Assurer la communication entre processus (sur des machines différentes
  - Ce sont des protocoles “de bout en bout” (pas de vision des sites intermédiaires)
  - Ce sont les protocoles visibles par les applications (les applications ne “voient” pas les protocoles de niveau inférieur)
- **Problèmes**
  - Un protocole de transport utilise un protocole de réseau (IP) qui
    - perd des messages (ou les délivre en double)
    - ne respecte pas l’ordre d’émission
    - limite la taille des messages
  - Un protocole de transport doit fournir des garanties aux applications utilisatrices
    - garantie de délivrance des messages
    - respect de l’ordre d’émission
    - pas de limitation de taille (flot de données)
    - synchronisation et contrôle de flux
- **Exemples**
  - UDP (minimal), TCP (avec garanties), RPC (intégré à un langage)

Identification des processus

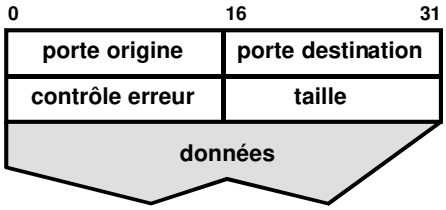
- **Problème**
  - Un protocole de transport fait communiquer deux processus (sur des hôtes différents). Il faut pouvoir identifier ces processus.
  - L’identification par le numéro interne (*pid* d’Unix ou équivalent) est inadéquate
    - elle est liée à un système d’exploitation particulier
    - elle identifie un processus individuel, alors qu’on souhaite plutôt identifier une classe de processus équivalents (rendant un service) ; un processus peut disparaître et être remplacé par un nouveau processus rendant le même service
- **Solution adoptée**
  - On utilise une identification indirecte au moyen de “portes”
  - Une porte est un point d’entrée prédéfini sur une machine, identifié par un numéro de porte (16 bits, soit 64 K portes possibles)
  - Un processus peut être associé à une porte (il “attend” derrière la porte)
  - Un message parvenant à une porte est reçu par le processus associé à la porte
  - Il existe des conventions d’usage des portes pour des services standard (typiquement les portes de numéro <1024)

un processus est identifié par un couple (adresse IP de l’hôte, numéro de porte)



Protocole UDP

- **Un protocole de transport "minimal"**
  - Fournit une simple transposition de IP au niveau transport ("IP de bout en bout")
  - Communication entre processus en mode non connecté (les messages sont indépendants les uns des autres)
  - Pas plus de garanties que IP
- **Format**



en-tête UDP

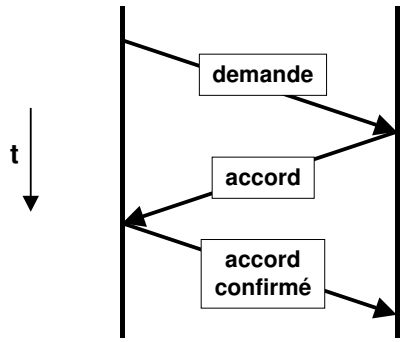
- **Propriétés**
  - Avantage : simplicité
  - Restrictions : garanties minimales
  - Usage : réalisation d'applications simples et peu exigeantes ; construction de protocoles plus élaborés

Protocole TCP

- **TCP permet de transmettre un "flot d'octets" bidirectionnel entre un processus origine (émetteur) et un processus destination (récepteur)**
- **Propriétés assurées**
  - Fiabilité (garantie de livraison dès lors qu'une connexion physique existe)
  - Préservation de l'ordre d'émission
  - Contrôle de flux (le récepteur peut demander à l'émetteur de réduire son débit d'émission pour éviter de dépasser la capacité d'absorption du récepteur)
  - Contrôle de congestion (limitation du débit d'émission de l'émetteur pour éviter de saturer le réseau)
  - Noter la différence entre *contrôle de flux* et *contrôle de congestion*
    - dans les deux cas, on agit sur le débit d'émission
    - contrôle de flux (de bout en bout) : conditionné par la capacité du *récepteur*
    - contrôle de congestion (sur le réseau) : conditionné par la capacité du *réseau* (liaisons et routeurs)

Protocole TCP

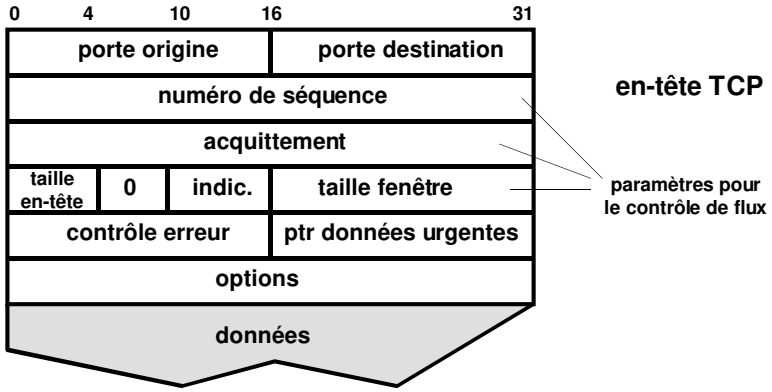
- **TCP fonctionne en "mode connecté"**
  - 3 phases
    - Phase de connexion : établir une liaison entre deux processus
    - Phase de communication : échanger des données entre les deux processus, sur la liaison établie
    - Phase de déconnexion : déconnecter les deux processus en supprimant la liaison
- **Connexion et déconnexion utilisent un mini-protocole (accord confirmé)**

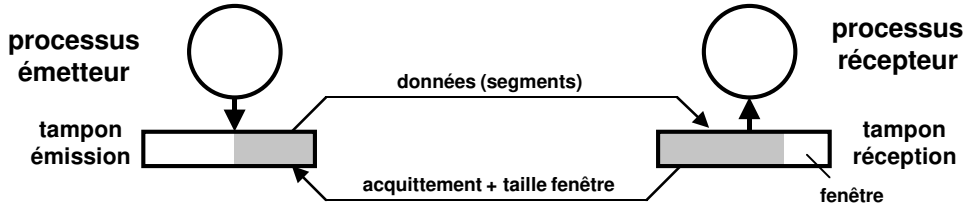


Les messages contiennent des paramètres permettant de fixer des caractéristiques de la liaison (pour la connexion)

Protocole TCP

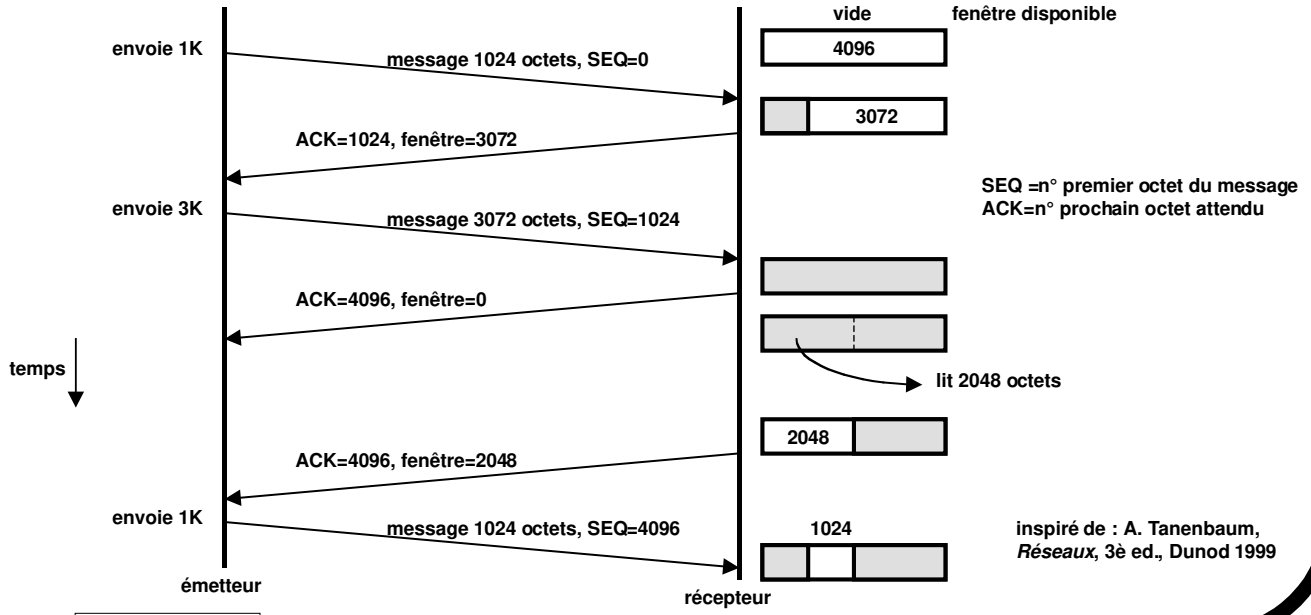
- **Format des données TCP**
  - Le flot d'octets (données) est transmis sous forme de tranches successives ("segments")





Protocole TCP : principe (simplifié) du contrôle de flux

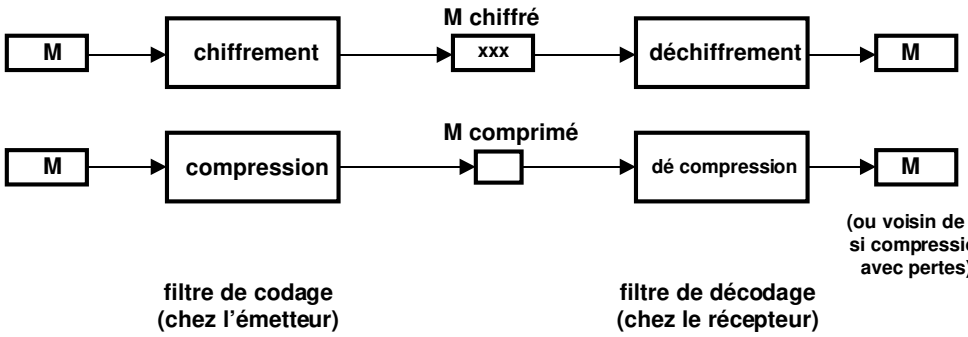
- Le récepteur accuse réception de chaque transmission en envoyant
  - le numéro du prochain octet attendu
  - la taille de la fenêtre de réception (capacité disponible pour recevoir des octets)
    - taille de fenêtre nulle = "arrêtez de m'envoyer des données"



inspiré de : A. Tanenbaum, *Réseaux*, 3è ed., Dunod 1999

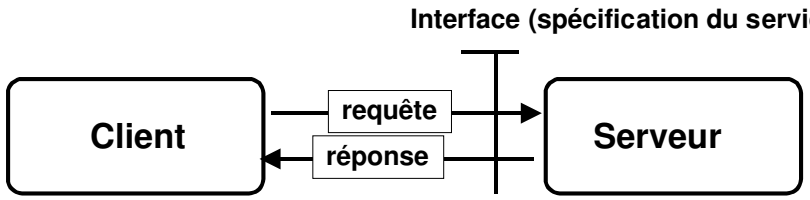
Protocoles de présentation

- **Deux aspects (importants !) dont nous ne parlons pas**
  - Sécurité
    - Essentiellement fondée sur la cryptographie (chiffrement des messages)
  - Compression de données
    - Nécessaire pour réduire l'encombrement en mémoire et surtout le temps de transmission des données de grande taille (multimédia)
  - Ces deux techniques (niveau "présentation" de l'ISO) utilisent des <u>filtres</u> (transformation des messages)



Applications : introduction au schéma Client-Serveur

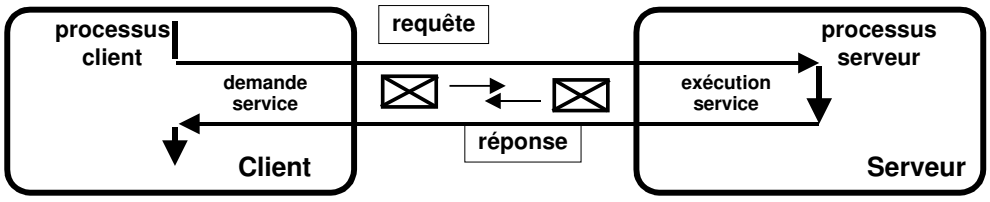
- **Principe**
  - Le *client* (un processus) demande l'exécution d'un service (spécifié par une interface)
  - Le *serveur* (un autre processus) réalise le service
  - Client et serveur sont (en général, pas nécessairement) localisés sur deux machines distinctes



- **Intérêt du schéma client-serveur**
  - Structuration
    - fonctions bien identifiées
    - séparation entre interface du service et réalisation (le client ne connaît que l'interface, qui définit le "contrat" entre client et serveur)
    - client et serveur peuvent être modifiés (remplacés) indépendamment
  - Protection
    - client et serveur s'exécutent dans des domaines de protection différents
  - Gestion des ressources
    - le serveur peut être partagé entre plusieurs clients

Schéma Client-Serveur

- **Client et serveur sont représentés par deux processus**
- **Les processus client et serveur communiquent par messages** (plutôt que par partage de données, mémoire ou fichiers)
  - Requête : paramètres d'appel, spécification du service requis
  - Réponse : résultats, indicateur éventuel d'exécution ou d'erreur
  - Communication *synchrone* (en général) : le processus client est bloqué en attente de la réponse



- **Identification des processus client et serveur**
  - Numéro de porte
  - Identification symbolique (voir RPC)

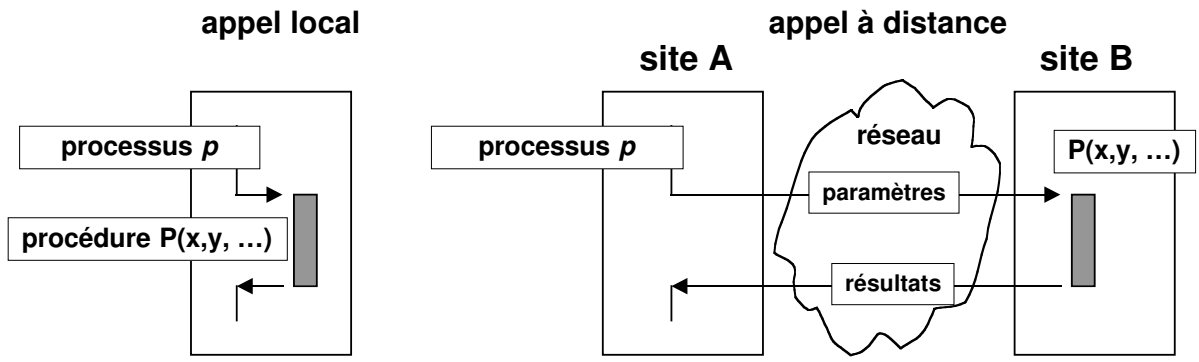
Mise en œuvre du schéma client-serveur

- **Principe**
  - Le schéma client-serveur est réalisé en utilisant les protocoles de transport ; cela peut être fait de deux manières
- **Opérations de "bas niveau"**
  - Utilisation de fonctions de communication fournies par le système d'exploitation et directement construites sur le protocole de transport
  - Exemple : *sockets* Unix [nous n'en parlons pas]
    - Mode non connecté (UDP)
    - Mode connecté (TCP)
- **Opérations de "haut niveau"**
  - Utilisation d 'un *middleware* spécialisé (logiciel d'interposition entre système de communication et applications)
  - Contexte : langage de programmation
    - Appel de procédure à distance [description séance 11]
  - Contexte : objets répartis
    - Appel de méthodes, création d'objets à distance [description séance 12]

Introduction à l'appel de procédure à distance (1)

Remote Procedure Call (RPC)

- **Outil de haut niveau pour la réalisation du schéma client-serveur**
- **Principe**



L'effet doit être identique dans les deux cas, vu de *p*.
Dans la pratique, ce n'est pas vrai en toute rigueur
en particulier en raison des erreurs possibles (perte de messages, panne d'un site

Introduction à l'appel de procédure à distance (2)

- **Avantages attendus**
  - Forme et effet identiques à ceux d'un appel local
    - Applications non modifiées lors du passage à l'appel distant
    - Mise au point locale avant répartition
    - Simplicité d'utilisation
  - Abstraction
    - Indépendance par rapport aux protocoles de communication
      - On n'a pas besoin d'apprendre un protocole de bas niveau
    - Réutilisation possible du code, y compris en environnement hétérogène
- **Problèmes**
  - Sémantique complexe en cas de panne
    - Les processus client et serveur peuvent tomber en panne indépendamment
    - Le réseau introduit une incertitude (pertes, retards)
  - Restrictions sur les paramètres
    - Passage de structures complexes, possible mais compliqué
- **Dans le cadre de ce cours**
  - Utilisation de l'appel de procédure à distance dans le cadre du langage Tcl

Résumé de la séance 9

- **Protocoles de transport**
  - Fonctions et problèmes
  - Protocoles UDP et TCP
  - L'interface système (processus, portes)
- **Introduction au protocole client-serveur**
  - Principe et organisation générale
  - Outils
- **Plan de la suite**
  - Prochaine séance : exemples concrets de systèmes client-serveur (DNS, Web, etc.)
  - Séance suivante : un outil pour les applications client-serveur, l'appel de procédure à distance ; mise en œuvre dans Tcl, exemples