Environnement Client/Serveur

Cours 2
Généralités sur les réseaux
IP
UDP

kn@lri.fr
http://www.lri.fr/~kn

UNIVERSITÉ PARIS SUD
Comprendre le monde, construire l'avenir

université PARIS-SACLAY

Plan

Définitions

Réseau
: ensemble de nœuds reliés entre eux par des liens (ou canaux).

Réseau informatique
: réseau où les nœuds sont des ordinateurs. Les liens sont hétérogènes (câbles, liaisons radio, liaisons satellites, ...)

Protocole
: ensemble de conventions permettant d'établir une communication mais qui ne font pas partie du sujet de la communication.

Organismes de Standardisation

Plusieurs organismes interviennent à différents niveaux

ISO
: *International Organisation for Standardisation*, standardise **TOUT**. Elle publie un standard pour les réseaux informatiques (le modèle Open System Interconnection).

IETF
: *Internet Engineering TaskForce*, organisme ouvert qui standardise les protocoles *TCP / IP*

IEEE
: *Institute of Electrical and Electronic Engineers Standard Association*, association originellement Américaine (maintenant internationale). Definit certains standards (ex: IEEE 802.11 a/n, ...).

W3C
: *World Wide Web Consortium* standardise les formats du Web comme HTML, SVG, XML, ...

Industrie
: les gros acteurs ont énormément d'influence (Google, Microsoft, Cisco, ...)

Les 7 couches du modèle OSI

N	Unité	Nom	Utilisation
7	—	*Application*	Logiciel
6	—	*Presentation*	Chiffrement
5	—	*Session*	Connexion, identification
4	Segment	*Transport*	Intégrité des données
3	Paquet	*Network*	Acheminement (routage)
2	Trame	*Data-link*	Encodage sur le support physique
1	Bit	*Physical*	Matériel (voltage, nature des câbles, ...)

Chaque couche règle de manière transparente pour les couches supérieures un problème spécifique. Une couche de niveau *N* n'a aucun contrôle sur une couche de niveau inférieur.

Ce modèle constitue la norme ISO/IEC 7498-1.

(ISO 7497 le engrais, ISO 7495 concerne les farines de blé tendre, ISO 7490 les implants dentaires, ...).

Plan

Modèle « TCP/IP »

Modèle en 4 couches, similaire au modèle OSI :

N	Nom	Description	Eq. OSI
4	*Application*	HTTP, Bitorrent, FTP, ...	5, 6, 7
3	*Transport*	TCP, UDP, SCTP, ...	4
2	*Internet*	IP (v4, v6), ICMP, IPsec, ...	3
1	*Link*	Ethernet, 802.11, ...	1, 2

TCP
: *Transfer Control Protocol*, s'occupe de la bonne transmission des données (détection d'erreurs, retransmission en cas de perte, ...)

UDP
: *User Datagram Protocol*, s'occupe de la bonne transmission des données (mais de manière plus basique que TCP)

IP
: *Internet Protocol*, s'occupe de deux choses :
- donner une adresse unique aux machines
- router les paquets, c'est à dire les transmettre à une autre machine, plus proche de

Structure d'un paquet IPv4

14 champs (13 obligatoires, 1 optionnel) :

	0—3	4—7	8—13	14—15	16—18	19—31
0	version	longueur en-tête	DSCP	ECN	longueur totale	
32	identification				flags	position du fragment
64	TTL		Protocole		Somme de contrôle	
96	Adresse IP de la source					
128	Adresse IP de la destination					
160	Options (si longueur en-tête > 5)					
> 160	Données					

Le paquet est transmis en *big endian* (c'est à dire les octets de poids fort en premier).

Détail des champs obligatoires

Version
: Numéro de version du protocole (4)

Longueur en-tête
: Longueur de l'en-tête en mots de 32bits (5 si pas d'option)

DSCP
: *Differentiated Service Code Point*, code du type de service (Voix sur IP, vidéo, jeu, message, ...)

ECN
: *Explicit Congestion Notification*, utilisé pour la gestion de l'engorgement

Longueur totale
: en octets, en-tête + données

Identification
: numéro servant à identifier le fragment

Flags
: bit 0 toujours à 0, bit 1 à 1 si le paquet ne doit pas être fragmenté, bit 2 à 0 si c'est le dernier paquet d'un ensemble fragmenté

Détail des champs obligatoires (suite)

Position du fragment
: position du fragment par rapport à un paquet non fragmenté

TTL
: *Time to Live* compteur décrémenté à chaque fois que le paquet traverse un nœud. Si 0 le paquet est jeté et une notification est envoyée à l'adresse de la source.

Protocole
: Code du protocole utilisé dans la partie données

Somme de contrôle
: nombre calcule à partir de l'en-tête, vérifiée à chaque retransmission

adresse source
: Adresse IP de l'envoyeur

adresse destination
: Adresse IP du receveur

Champs optionnels

Les champs optionnels commencent par un octet d'identification. Plusieurs options possibles :

Routage spécifié
: Liste de tous les nœuds à suivre pour arriver à la destination

Routage large
: Liste de nœuds que le paquet doit traverse

Enregistrement de route
: Liste de nœuds qui ont été traversé par le paquet

...

Calcul de la somme de contrôle

Le calcul de somme de contrôle est une opération très courante en informatique, elle permet de détecter certaines corruptions de données.

- Calcul de la somme de contrôle :
  1. On découpe l'en-tête en nombres de 16 bits en excluant le champ de somme de contrôle et on appelle `N` la somme de tous ces nombres
  2. Tant que `N >= 65536`, `N := (N mod 65536) + (N / 65536)`
  3. On représente le nombre `N` en binaire puis on inverse les 1 et les 0
  4. On écrit le mot de 16 bits résultant dans le champ *somme de contrôle*
- Vérification de la somme de contrôle :
  1. On découpe l'en-tête en nombres de 16 bits en incluant le champ de somme de contrôle et on appelle `N` la somme de tous ces nombres
  2. Tant que `N >= 65536`, `N := (N mod 65536) + (N / 65536)`
  3. On représente le nombre `N` en binaire puis on inverse les 1 et les 0
  4. Si on obtient 0, la vérification est correcte, sinon le paquet est corrompu.

Plan

Adresse IP (v4)

- Moyen d'identifier une machine sur Internet
- Adresse composée de 4 octets (32 bits) :

ex: 129.175.28.179

- Certaines adresses sont réservées : 10.x.x.x, 172.[16—31].x.x, 192.168.x.x, 169.254.x.x
- *Internet Assigned Numbers Authority*, attribue les IPs par bloc aux fournisseurs d'accès et grandes entreprises
- IPv4 est amené à disparaître, remplacé par IPv6 (adresses sur 128 bits)

Classes d'adresses IP

Une adresse IP se décompose en deux parties :

1. Une partie réseau (assignée par l'IANA)
2. Une partie machine (assignée par l'administrateur réseau local)

Il y a 5 classes d'adresses (3 utilisées en pratique). On note N la partie réseau et H la partie machine :

classe A
    [0—127].H.H.H
classe B
    [128—191].N.H.H
classe C
    [192—223].N.N.H

Deux classes spéciales :

classe D
    [224—247].X.X.X, multicast

Nombre de machines adressables

classe A
    [0—127].H.H.H, 7 bits pour le réseau, 3 octets pour la machine
classe B
    [128—191].N.H.H, 14 bits pour le réseau, 2 octets pour la machine
    ex: 129.175.125.111
classe C
    [192—223].N.N.H, 5 bits plus 2 octets pour le réseau, 1 octet pour la machine

Lors d'un adressage par classe, une machine ne peut communiquer directement qu'avec d'autres machines du même réseau

Adresses réservées

Certaines adresses ont une utilisation particulière et ne peuvent pas être attribuées à des machines :

La machine locale
: 127.0.0.1 (permet à une machine d'avoir une adresse réseau même si elle n'est pas connectée)

Tout le réseau
: On remplace la partie machine par des 0 (par exemple pour une classe B : 129.175.0.0

Diffusion
: On remplace la partie machine par des 255 (par exemple pour une classe B : 129.175.255.255). Permet d'envoyer un message à toutes les machines du réseau en même temps.

Routage

Une machine A veut envoyer un message à une machine D. Les deux machines ne sont pas *directement reliées* (pas de câble point-à-point, pas sur le même *switch*, pas sur le même réseau Wifi, ...). Différentes couches matérielles (i.e. RJ45 et Wifi) ne savent pas parler entre elles.

Routeur
: machine possédant au moins 2 interfaces (périphériques) réseau ainsi que des logiciels spécialisés et dont le but est de transmettre les paquets IP d'un réseau vers un autre

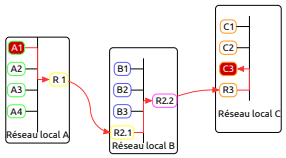
Table de routage
: spécifie, pour chaque groupe d'IPs, quel est le routeur

Exemple :

```
$ route
Kernel IP routing table
Destination     Gateway         Genmask         ... Iface
192.168.0.0     *               255.255.255.0   ... eth1
129.175.240.0   *               255.255.255.0   ... eth0
default         129.175.240.1   0.0.0.0         ... eth0
```


Routage (suite)

Transmission d'un paquet d'une machine A1 à une machine C3



Épuisement des adresses IPv4

Combien de machines peut on adresser avec IPv4 ?

$256 \times 256 \times 256 \times 256 = 2^{32} = 4\ 294\ 967\ 296$

Insuffisance du nombre d'adresses liée aux facteurs suivants :

- Explosion du nombre de terminaux mobiles (on a dépassé le milliard en 2012)
- Internet of Things (montres, voitures, chaussures, ... connectées).
- Connexions permanentes (machines « toujours allumées et connectées »)
- Adressage par classe inadapté (rien entre les 16 millions d'adresses d'une classe A et les 65536 adresses d'une classe B). Si une entité veut 1 million de machines adressables, il lui faut 15 classes B.

Quelles solutions ?

IPv6
- Nouvelle version du protocole IP :
- avantage
  - adresses sur 128 bits ($2^{128} = 3.4 \times 10^{38}$)
- inconvénient
  - standard incompatible avec IPv4

Adressage par sous-réseau
- on re-découpe la partie « machine » d'une adresse pour dénoter un sous-réseau

Réseaux locaux
- certaines machines sont « cachées » du réseau global et utilisées uniquement en interne

Mise en sous-réseau

On utilise un masque *i.e.*un nombre dont le « et » binaire avec l'adresse va isoler la partie réseau.

- adresse : 129.175.34.35 (classe B)
- masque : 255.255.240.0 (= $11111111.11111111.11110000.00000000_2$)
- Application du masque

décimal	129	175	34	35
binaire	10000001	10101111	00100010	00100011
masque	11111111	11111111	11110000	00000000
« ET »	10000001	10101111	00100000	00000000
sous-réseau	129	175	32	0
masque inversé	00000000	00000000	00001111	11111111
« ET »	00000000	00000000	00000010	00100011
machine	0	0	2	35

On peut donc, au moyen d'un masque, découper une classe (A par exemple) en sous-réseaux plus petits

Réseaux privés

Le standard réserve 3 blocs d'adresses :

- 10.0.0.0 — 10.255.255.255
- 172.16.0.0 — 172.31.255.255
- 192.168.0.0 — 192.168.255.255

Ces adresses ne peuvent jamais être données à une machine sur Internet. Elles sont utilisées par le réseau local. Seul le routeur possède une adresse publique et il s'occupe de faire la traduction entre adresses locales et adresses globales (NAT : Network Address Translation)

Port

La couche de transport (TCP ou UDP) définit une notion de port. Un port est un identifiant numérique associé à une connexion. Un service (ou serveur) est identifié de manière unique par son adresse IP et son numéro de port. Par convention, certains ports sont réservés pour des services particuliers :

- TCP/21 : FTP
- TCP/22 : SSH
- TCP ou UDP/53 : DNS
- TCP/80 : HTTP
- TCP/110 : POP
- TCP/443 : HTTPS
- UDP/5353 : MDNS

L'utilité d'un port est de pouvoir faire « tourner » plusieurs services sur la même machine (même IP).

Pare-feu

Un pare-feu (firewall) est simplement un routeur qui applique une politique de sécurité. L'administrateur définit un ensemble de règles qui disent au pare-feu quels paquets il doit transmettre et quels paquets il doit arrêter. Les règles peuvent se baser sur :

- L'adresse source ou destination
- le numéro de port
- le contenu du paquet
- ...

DNS

Domain Name System : permet d'attribuer un nom à une IP (annuaire). Double avantage :

- pour les humains, un nom est plus simple à retenir
- on peut changer d'adresse IP de manière silencieuse

Principe hiérarchise :

- les serveurs DNS primaires gardent les informations sur les TLD (Top Level Domain : .com, .fr, .net, ...)
- pour chaque tld, il y a un ensemble de serveur DNS de niveau 2 qui fait correspondre le nom de domaine (google.com, u-psud.fr) à un DNS de niveau 3 (généralement le DNS de niveau 2 est chez le FAI)
- le DNS de niveau 3 donne l'IP d'une machine particulière sur son domaine : mail, www (le DNS de niveau 3 est administré localement)

Adresses en Java (1)

En Java, la classe `java.net.InetAddress` propose une abstraction pour la résolution de noms et les adresses IP (v4 aussi bien que v6). La Javadoc fournit une description complète de ce qu'est une adresse IP et des facilités fournies par la classe `InetAddress`. La classe *ne possède pas de constructeur public*. On peut créer des adresses au moyen des méthodes statiques suivantes :

`getByName(String host)` :
: une chaîne de caractères représentant l'hôte. Cela peut être son nom (implique une résolution DNS), son adresse IPv4 ou son adresse IPv6.

`getAllByName(String host)` :
: comme précédemment mais renvoie un tableau de toutes les adresses connues pour l'hôte donné.

`getByAddress(byte[] addr)` :
: renvoie l'addresse correspondant aux 4 (IPv4) ou 16 (IPv6) octets placés dans le table `addr`. Rappel : on appelle les méthodes statiques directement sur la classe pas sur un objet particulier : `InetAddress.getAllByName("www.google.com");`

Plan

Le protocole UDP

Le protocole UDP (*User Datagram Protocol*) est un protocole de la couche de transport, donc au dessus d'IP. Son but est de permettre d'envoyer des messages entre deux hôtes identifiés par leurs adresses IP ou entre un hôte et un ensembles de cibles (multicast). Les messages envoyés sont appelés des *datagrammes*.
UDP est un protocole sans connexion. Cela signifie que les datagrammes sont routés et envoyés sans qu'il y ait de notion de séquence d'ensemble de messages. C'est un protocole très simple, orienté performances. En particulier, il n'y a pas de négociation (*handshake*) entre l'envoyeur et le receveur :

- Les datagrammes peuvent êtres perdus sans notification
- les datagrammes peuvent arriver dans n'importe quel ordre
- les datagrammes peuvent arriver en plusieurs exemplaires

Intérêt du protocole UDP

Le protocole UDP est orienté performance et simplicité. Il est particulièrement approprié dans les cas suivants :

Envois temps réel
: l'absence de retransmission en cas d'erreur permet d'attendre des performances temps réel (désirable par exemple pour les jeux en réseau)

Mise en place de connexion
: envoie de paquets simple pour établir un protocole plus complexe (via TCP), par exemple DHCP qui permet d'assigner dynamiquement des adresses IP

Diffusion large
: si un même serveur doit être connecté à un nombre important de clients (par exemple diffusion de vidéo en streaming)

Établissement de connexion, façon POSIX

Le modèle POSIX expose un type de donnée abstrait appelé *socket* (prise). Une *socket* est indépendante de la couche de transport (TCP, UDP, autre) ainsi que du mode d'adressage (IP v4, v6, autre). Une *socket* peut être utilisée de deux manière différente :

1. En mode « écoute » (*bind*) : c'est le mode privilégié pour le serveur
2. En mode « connecté » (*connected*) : c'est le mode privilégié pour le client

Mode écoute

Ce mode permet d'associer une socket à une ou plusieurs adresses (elles mêmes associées à des périphériques sous-jacent, la plupart du temps des cartes réseau) et sur un port particulier.

Une fois la socket en mode écoute, on peut envoyer et recevoir des messages selon les spécificités du protocole considéré (TCP, UDP, ...)

Mode connexion

Ce mode permet d'associer une socket à une adresse particulière (celle du serveur) et à un port particulier.

Une fois la socket en mode connexion, on peut envoyer et recevoir des messages selon les spécificités du protocole considéré (TCP, UDP, ...)

Architecture générale des programmes

Le serveur crée une socket en mode écoute, sur une ou plusieurs adresses, et sur un port particulier. L'information du port fait partie du protocole et doit être connu des clients, comme les adresses. Le serveur se met ensuite à attendre des messages sur sa socket. Pour chaque message, il connait :

- L'adresse de l'envoyeur
- Le port de l'envoyeur
- Le contenu du message

Sur réception d'un message, le serveur traite le message puis repasse en attente d'un nouveau message

Le client crée une socket en mode connexion vers une adresse (ou un nom) et un port particuliers. Il peut ensuite envoyer des messages au serveur ou attendre des messages de ce dernier.

UDP en Java

Les classes permettant d'écrire des applications Client/Serveur utilisant le protocole UDP en Java sont :

`DatagramSocket` :
: Représente une socket pour une connexion UDP. On peut la placer en mode écoute ou connexion

`DatagramPacket` :
: Représente un datagramme, c'est à dire un message envoyé ou reçu via une socket

La classe `DatagramSocket`

On crée un objet d'une telle classe via son constructeur. On peut passer beaucoup de paramètres distincts. Pour le cours, on ne passera que `null` (pour avoir les paramètres par défaut). Une fois un tel objet crée on a accès aux méthodes suivantes :

`.bind(InetSocketAddress addr)` :
: Place la socket en mode écoute sur l'adresse de socket donnée. Peut créer une adresse d'écoute de la manière suivante :
  ```
  new InetSocketAddress(port);
  ```
  où `port` est un numéro de port sur lequel écouté. Une telle adresse est associée par défaut à tous les périphériques réseau de la machine.

`.connect(InettAddress addr, int port)` :
: place la socket en mode connexion sur l'adresse donnée et le port donné.

`.send(DatagramPacket p)` :
: Envoie le message contenu dans le datagramme `p` à l'autre bout de la socket.

`.receive(DatagramPacket p)` :
: Reçoit une message sur la socket et stocke son contenu dans le datagramme `p`.

La méthode `receive`

La méthode receive est par défaut blocante. Le programme attend de recevoir un message pour continuer son exécution. Comme UDP ne maintient pas d'information de connexion, s'il n'y a aucun client connecté ou que les paquets sont perdus, le serveur attend pour toujours ! On peut utiliser un *timeout* pour gérer ce problème. La méthode `setSoTimeout(int t)` permet de placer sur une socket un timeout de `t` millisecondes. Si aucun message n'est reçu au bout de `t` millisecondes, l'exception `SocketTimeoutException` est levée.

La classe `DatagramPacket`

Cette classe représente les messages envoyés sur la socket. On crée un objet de la manière suivante :

```
DatagramPacket p = new DatagramPacket (bytes, len);
```

où `bytes` est un tableau de `bytes` et `len` la taille à utiliser dans ce tableau. Lorsque l'on passe un tel objet à la méthode `receive`, cette dernière rempli le tableau avec le message reçu, jusqu'à la longueur `len`. Si le message reçu est plus grand, alors les octets supplémentaires sont perdus, sans avertissement. On peut connaître la zone du tableau qui a été remplie par en utilisant `.getOffset()` (indice initial) et `.getLength()` (longueur de la zone occupée).

Exemple de code serveur

```
//Ouverture d'une socket en écoute sur le port UDP 10000.
DatagramSocket s = new DatagramSocket(null);
s.bind(new InetSocketAddress(10000));
//Timeout de 1 seconde
s.setSoTimeout(1000);
//Création d'un tableau de 256 octets puis d'un datagramme associé
byte [] buff = new byte[256];
DatagramPacket p = new DatagramPacket(buff, 256);
while (true)
{
  try {
   s.receive(p);    //reception

   System.out.println("Reçu: " + new String(p.getData(), 0, p.getPos()));

  } catch (SocketTimeoutException e) {
   System.out.println("Pas de message reçu pendant 1 seconde");
  }
}
```


Exemple de code client

```
//Ouverture d'une socket en connexion sur le port UDP 10000
//du serveur 'serveur.com'
DatagramSocket s = new DatagramSocket(null);
s.connect(InetAddress.getByName("serveur.com"),10000);

//Construction d'un message à partir d'un chaîne
String message = "Hello, Server !";
byte[] buff = message.getBytes();

s.send(new DatagramPacket(buff, buff.length));

//Attention, si le serveur attend moins de caractères,
//les caractères en plus seront perdus.
```