Architecture des réseaux et des ordinateurs

Alexandre Guitton

L3 informatique

Organisation du cours

- 15h CM + 18h TP
- 3 crédits
- Contrôles :
  - Session 1 : TP noté (30%) + examen (70%)
  - Session 2 : examen (100%)

Plan du cours

- Introduction aux réseaux et modèles des réseaux en couches
- Notions de base sur IP (et ICMP et ARP), UDP, TCP et DNS
- Protocoles web : HTTP et FTP
- Protocoles email : SMTP, POP, IMAP
- En TP : programmation socket et manipulation d’outils réseaux

Chapitre 1 – Introduction aux réseaux

- Notions introduites dans ce chapitre :
  - Les sept couches du modèle OSI
  - Les quatre couches du modèle Internet

Principes

- Les réseaux sont constitués de plusieurs machines distantes
  - Problématique d'interopérabilité => normalisation des communications (standards, RFC, etc.)
- Les communications réseaux impliquent beaucoup de mécanismes
  - Création de nombreux protocoles
  - Les protocoles sont regroupés par fonctionnalités dans des couches
  - Les données sont encapsulées (ex : les données HTML sont encapsulées dans des données HTTP, qui sont encapsulées dans un paquet TCP, qui sont encapsulées dans un paquet IP, qui sont encapsulées dans une trame Ethernet)

Modèle OSI

- OSI = Open Systems Interconnection
  - Proposition en 1978, normalisation en 1984
  - Normalisé par l'ISO (International Standard Organization)
- Sept couches :
  - Physique (manipule des bits)
  - Liaison de données (manipule des trames)
  - Réseau (manipule des paquets)
  - Transport (manipule des segments)
  - Session (manipule des données)
  - Présentation (manipule des données)
  - Application (manipule des données)

Couches basses

- Couche physique : transmission de bits
  - Modulation = transformation d'un signal binaire en un signal adapté au médium (ex : sinusoïde paramétrée avec amplitude / phase / fréquence)
  - Codage = manière dont les bits sont encodés (ex : codage de Manchester, codage de Manchester différentiel, codage NRZ, etc.)
- Couche liaison de données : adressage physique et partage du temps de parole
  - Détection et correction d'erreur, retransmission
  - Tour de rôle, accès multiple avec contention

Couches intermédiaires

- Couche réseau : adressage logique, routage et acheminement
  - Routage = calculer la route à suivre de la source à la destination
  - Acheminement = suivre la route calculée
- Couche transport : communication de bout en bout
  - Multiplexage

Couches hautes

- Couche session : gère la reconnexion automatique
- Couche présentation : encodage des données
  - Codage
  - Compression
  - Chiffrement
- Couche application : non spécifiée

Modèle Internet (ou modèle TCP/IP)

- Conçu en 1976, mais non normalisé
- Quatre couches :
  - Couche accès réseau (= couche physique + couche liaison de données)
  - Couche Internet (= couche réseau)
  - Couche transport (= couche transport)
  - Couche application (= couche session + couche présentation + couche application)

Chapitre 2 – Protocoles Internet de base

- Notions introduites dans ce chapitre :
  - Fonctionnement d'IP, ICMP et ARP
  - Fonctionnement d'UDP (et programmation socket UDP)
  - Fonctionnement de TCP (et programmation socket TCP)
  - Fonctionnement de DNS

Fonctionnement d’IP (#1)

- IP = Internet Protocol (1981)
- Fonctionnalités principales : adressage et acheminement
  - Pas de routage : rôle des protocoles de routage (RIP, OSPF, BGP, etc.)
- Adressage
  - IPv4 = 32 bits, IPv6 = 128 bits
  - Notation à points
- Adressage IPv4
  - Classes A, B et C (et D et E)
  - Masques de réseaux, adresse de réseaux, notation « /x » pour les x premiers bits à 1 dans le masque
  - CIDR (= Classless Inter-Domain Routing)

Fonctionnement d’IP (#2)

- Acheminement = longest common prefix match
  - Notion de route par défaut
  - Notion de passerelle (plutôt pour les équipements terminaux)
- Autres fonctionnalités : fragmentation, détection d’erreurs

Format de l’entête IP

Version (4 bits)	Longueur entête sur 4	Type de service (8 bits)	Longueur totale	
Identification du paquet			Flags (3 b.)	Position du fragment (13 bits)
TTL		Protocole encapsulé	Somme de contrôle	
Adresse IP source (32 bits)				
Adresse IP destination (32 bits)				

Fonctionnement d’ICMP

- ICMP = Internet Control Message Protocol (1981)
- Fonctionnalité : gestion des messages de contrôle d’IP
- Types de messages ICMP :
  - Echo request et echo reply (ping)
  - Destination unreachable
  - TTL exceeded
  - Timestamp request et timestamp reply
  - Etc.

Fonctionnement d’ARP

- ARP = Address Resolution Protocol (1982)
- Fonctionnalité : donner l’adresse MAC d’une machine du réseau dont l’adresse IP est connue
  - Utilisé pour l’acheminement de proche en proche

Fonctionnement d'UDP

- UDP = User Datagram Protocol (1980)
- Fonctionnalités principales :
  - Multiplexage, au travers de ports (= numéros servant à identifier le service (côté serveur) ou l'application source (côté client))
- Format de l'entête UDP

Port source (16 bits)	Port destination (16 bits)
Longueur	Somme de contrôle

Programmation de sockets UDP

Serveur UDP

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define MAX 100
#define SERVER_PORT 10000

int main(void) {
        int s;
        char buffer[MAX];
        struct sockaddr_in server, client;
        s = socket(PF_INET, SOCK_DGRAM, 0);
        server.sin_family = AF_INET;
        server.sin_addr.s_addr = INADDR_ANY;
        server.sin_port = htons(SERVER_PORT);
        bind(s, (struct sockaddr *)&server, sizeof(struct sockaddr));
        recvfrom(s, buffer, MAX, 0, (struct sockaddr *)&client, NULL);
        printf("message=%s\n", buffer);
        close(s);
        return 0;
}
```

Client UDP

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#define MAX 100
#define SERVER_PORT 10000
#define SERVER_ADDRESS "192.168.1.1"

int main(void) {
        int s;
        char * message = "Hello world";
        struct sockaddr_in server;
        s = socket(PF_INET, SOCK_DGRAM, 0);
        server.sin_family = AF_INET;
        server.sin_addr.s_addr = inet_addr(SERVER_ADDRESS);
        server.sin_port = htons(SERVER_PORT);
        sendto(s, message, strlen(message), 0,
                (struct sockaddr *)&server, sizeof(struct sockaddr));
        close(s);
        return 0;
}
```

À savoir avec UDP…

- La création d'un client/serveur UDP est facile a priori, car il y a peu d'appels systèmes
  - Mais, il faut gérer les pertes éventuelles de paquets, la fragmentation, etc.
- UDP est un protocole rapide (autant qu'IP) mais non fiable (pas plus qu'IP)
  - UDP est adapté aux communications temps-réel (visio-conférence, audio-conférence, jeux, etc.)

Fonctionnement de TCP

- TCP = Transmission Control Protocol (1980)
- Fonctionnalités principales :
  - Multiplexage, au travers de ports
  - Mode connecté : garantie de livraison + non duplication + ordonnancement
  - Contrôle de flux, contrôle de congestion

Format de l’entête TCP

<table>
<tr><td colspan="3">Port source (16 bits)</td><td>Port destination (16 bits)</td></tr>
<tr><td colspan="4">Numéro de séquence (32 bits)</td></tr>
<tr><td colspan="4">Numéro d’acquittement (32 bits)</td></tr>
<tr><td>Longueur entête sur 4 (4 bits)</td><td>Réservé (6 bits)</td><td>Flags (7 bits)</td><td>Taille de la fenêtre</td></tr>
<tr><td colspan="3">Somme de contrôle</td><td>Pointeur sur les données urgentes</td></tr>
<tr><td colspan="4">Options éventuelles (0 bits ou plus)</td></tr>
</table>

Programmation de sockets TCP

Serveur TCP

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>

#define MAX 100
#define SERVER_PORT 10000

int main(void) {
        int s1, s2, length;
        char buffer[MAX];
        struct sockaddr_in server, client;
        s1 = socket(PF_INET, SOCK_STREAM, 0);
        server.sin_family = AF_INET;
        server.sin_addr.s_addr = INADDR_ANY;
        server.sin_port = htons(SERVER_PORT);
        bind(s1, (struct sockaddr *)&server, sizeof(struct sockaddr));
        listen(s1, 5);
        s2 = accept(s1, (struct sockaddr *)&client, &length);
        recv(s2, buffer, MAX, 0);
        printf("message=%s\n", buffer);
        close(s2);
        close(s1);
        return 0;
}
```

Client TCP

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#define MAX 100
#define SERVER_PORT 10000
#define SERVER_ADDRESS "192.168.1.1"

int main(void) {
        int s;
        char * message = "Hello world";
        struct sockaddr_in server;
        s = socket(PF_INET, SOCK_STREAM, 0);
        server.sin_family = AF_INET;
        server.sin_addr.s_addr = inet_addr(SERVER_ADDRESS);
        server.sin_port = htons(SERVER_PORT);
        connect(s, (struct sockaddr *)&server, sizeof(struct sockaddr));
        send(s, buffer, strlen(buffer)+1, 0);
        close(s);
        return 0;
}
```

À savoir avec TCP

- La création d'un client/serveur TCP est un peu plus compliqué a priori qu'avec TCP
  - Mais, le développement du reste de l'application est beaucoup plus simple
- TCP est un protocole lent (par rapport à IP et UDP) mais fiable (contrairement à IP ou UDP)
  - TCP est adapté aux communications fiables (transfert de fichier, transfert de texte)

Fonctionnement de DNS (#1)

- DNS = Domain Name System (1983)
  - Utilise le port 53 (en TCP et en UDP)
- Fonctionnalité principale : traduction de noms en adresses IP
  - Base de données distribuée (de manière hiérarchique) fournissant un service d'annuaire
- Enregistrements DNS
  - Adresses IPv4 (A) et IPv6 (AAAA), alias (CNAME)
  - Serveurs de noms (NS)
  - Serveurs SMTP (MX)
  - Etc.

Fonctionnement de DNS (#2)

- Espace de noms = structure arborescente
  - Chaque nœud a un nom et des enregistrements associés
  - La descente dans l'arbre se fait en ajoutant un « . »
  - Un nœud peut déléguer la responsabilité d'une zone à un serveur de noms
  - Un domaine est un chemin, ou un nom lorsqu'il n'y a pas d'ambiguïté
- Mécanisme de résolution d'adresses
  - Un résolveur détermine le serveur de noms responsable d'un domaine en suivant le chemin dans l'arbre, en partant de la racine
  - Des serveurs caches stockent ces informations temporairement
  - Les serveurs caches implémentent souvent eux-mêmes les requêtes récursives

Fonctionnement de DNS (#3)

- Gestion des dépendances circulaires : enregistrements « glue » (glue records)
  - Exemple : Le serveur de noms du domaine « isima.fr. » est « ns.isima.fr. ». Pour connaître l'adresse IP de « www.isima.fr. », un utilisateur obtient le nom « ns.isima.fr. », qu'il doit résoudre. Mais pour résoudre « ns.isima.fr. », qui est dans la zone « isima.fr. », il faut résoudre « ns.isima.fr ». On a une dépendance circulaire.
  - Les dépendances circulaires sont évitées par les enregistrements glue : l'adresse IP du serveur de noms est donnée dans la réponse.

Chapitre 3 – Protocoles applicatifs classiques

- Notions introduites dans ce chapitre :
  - Protocole HTTP
  - Protocole FTP
  - Protocole SMTP
  - Protocole POP
  - Protocole IMAP

Le protocole HTTP

- HTTP = HyperText Transfer Protocol (1996)
  - Utilise le port 80 (parfois 8000 ou 8080)
  - Un hypertexte est un texte pouvant contenir des liens vers d'autres ressources (incluant d'autres textes)
- Fonctionnalité principale : transférer des fichiers hypertextes
  - Protocole synchronisé de type requête-réponse
  - Un client demande un accès à des ressources HTTP, en utilisant des URL (Uniform Resource Locators). Les URL se décomposent en une adresse de serveur et une URI (Uniform Resource Identifier).
  - HTTP permet de transférer des fichiers textes (généralement au format HTML) ou des fichiers binaires (images, sons, vidéos)

Le protocole HTTP – les requêtes (#1)

- Format des requêtes
  - Une ligne contenant la requête (avec la méthode)
  - Des lignes d'entête de requête (optionnelles)
  - Une ligne vide
  - Un corps éventuel (optionnel)
- Méthodes de requêtes
  - GET : permet de demander une ressource
  - POST : pour soumettre une ressource (ou un formulaire)
  - OPTIONS : pour obtenir les méthodes supportées par le serveur pour une certaine URI
  - Etc.

Le protocole HTTP – les requêtes (#2)

- Entêtes de requêtes :
  - Accept (exemple : text/html)
  - Accept-Encoding (exemple : gzip)
  - Cookie (voir plus loin)
  - Host (utile en cas de proxy, obligatoire depuis HTTP 1.1)
  - User-Agent

Le protocole HTTP – les réponses (#1)

- Format des réponses :
  - Une ligne contenant le code de réponse et un message
  - Des lignes d'entête de réponse (optionnelles)
  - Une ligne vide
  - Un corps de réponse (optionnel)
- Codes de réponse :
  - 1XX = Information
  - 2XX = Succès
  - 3XX = Redirection
  - 4XX = Erreur client
  - 5XX = Erreur serveur

Le protocole HTTP – les réponses (#2)

- Entêtes de réponse :
  - Content-Encoding (exemple : gzip)
  - Content-Language (exemple : fr, en)
  - Content-Length
  - Content-Type (exemple : text/html; charset=utf-8)
  - Date
  - Expires
  - Location (en cas de redirection)

Le protocole HTTP – les cookies

- Certaines applications HTTP nécessitent de conserver un état
  - Exemple : une information d'authentification, un panier d'achats
  - Le serveur stocke associe les informations de connexion à une chaîne de caractères représentant la session de l'utilisateur
  - Le client stocke le numéro de la session dans un fichier texte, appelé cookie, et transfère ce numéro de session à chaque requête

Le protocole HTTP – exemple

- Exemple :

```
GET / HTTP/1.0
Host: www.isima.fr
User-Agent: Lynx
```

```
HTTP/1.0 200 OK
Server: Apache (Unix)
Content-Length: 97
Content-Type: text/html; charset=UTF-8
Set-cookie: SESSION_ID=123

<html>
<head><title>ISIMA</title>
<body>Bienvenue</body>
</html>
```

Le protocole FTP (#1)

- FTP = File Transfer Protocol (1985)
- Fonctionnalité principale : transfert de fichiers
  - Connection séparée pour les commandes (port 21) et les données (généralement port 20)
  - Dans le mode actif (par défaut), le client informe le serveur du port sur lequel il écoute pour recevoir les données (avec la commande PORT)
  - Dans le mode passif (déclenché avec la commande PASV), c'est le serveur qui informe le client du port choisi

Le protocole FTP (#2)

- Liste des commandes du client
  - USER et PASS pour le login et le password
  - CWD et CDUP pour changer de répertoire (descendre ou monter)
  - DELE pour effacer un fichier
  - HELP pour l'aide
  - LIST et NLST pour les informations sur le répertoire ou le fichier
  - PASV et LPSV pour entrer en mode passif (court ou long)
  - PORT pour informer de l'adresse et du port pour le transfert de données
  - PWD pour connaître le répertoire actuel
  - RETR pour télécharger un fichier
  - TYPE pour changer le mode de transfert (ASCII ou binaire)

Le protocole FTP (#3)

- Liste des codes de réponse du serveur (#1)
  - 1XX : réponse positive (début + milieu)
  - 2XX : réponse positive (fin)
  - 3XX : réponse positive en cours (en attente d'une autre requête)
  - 4XX : réponse négative transitoire
  - 5XX : réponse négative permanente
  - 6XX : réponse protégée

Le protocole FTP (#4)

- Liste des codes de réponse du serveur (#2)
  - X0X : syntaxe
  - X1X : information
  - X2X : connection
  - X3X : authentification
  - X5X : système de fichiers

Le protocole FTP – exemple

- Exemple :

```
ftp > open ftp.isima.fr
220- Welcome to this fictious FTP server.
220- FTP server ready.
Name: toto
331 Password required
Password:
230 User logged in.
ftp> dir
200 PORT command successful.
150 Opening ASCII mode data connection
for /bin/ls
drwx--x--x 1000 images
226 Transfer complete.
```

```
ftp> cd images
250 CWD command successful.
ftp> dir
200 PORT command successful.
150 Opening ASCII mode data connection
for /bin/ls
-rw-rw-r– 1000 toto.gif
226 Transfer complete.
ftp> bin
200 Type set to I.
ftp> get toto.gif
200 PORT command successful.
150 Opening BINARY mode data connection
for ex.gif
226 Transfer complete.
ftp> close
221 Goodbye.
ftp>
```

Le protocole SMTP

- SMTP = Simple Mail Transfer Protocol (1982), port 25
- Fonctionnalité : transmission d'emails (mais pas la consultation d'emails)
- Commandes du client SMTP
  - Commande MAIL : informe de l'adresse de l'émetteur
  - Commande RCPT : informe de l'adresse du destinataire
  - Commande DATA : informe du contenu du message (terminé par une ligne contenant uniquement « . »)
- Réponses du serveur SMTP
  - 2XX (réponse positive), 4XX (réponse négative transitoire) ou 5XX (réponse négative permanente)

Le protocole SMTP – exemple

- Exemple :

```
Server:   220 smtp.isima.fr Welcome
Client:   HELO alice
Server:   250 Welcome alice
Client:   MAIL FROM: <alice@isima.fr>
Server:   250 Ok
Client:   RCPT TO: <bob@isima.fr>
Server:   250 Ok
Client:   DATA
Server:   354 Finish data with \r\n.\r\n
```

```
Client:   From: « Alice Dupont »
Client:   To: « Bob Durand »
Client:   Subject: Test
Client:   Hello Bob,
Client:   Nice too meet you.
Client:   Alice
Client:   .
Server:   250 Ok
Client:   QUIT
Server:   221 Bye
```

Le protocole POP3

- POP3 = Post Office Protocol version 3 (1988), port 110
- Fonctionnalité : récupération/consultation d'emails
  - Protocole requête-réponse très simple
  - Les messages sont a priori transférés du serveur vers l'ordinateur de l'utilisateur

Le protocole POP3 – exemple

- Exemple :

```
Server:   +OK POP3 server ready
Client:   USER toto
Server:   +OK User accepted
Client:   PASS titi
Server:   +OK Pass accepted
Client:   STAT
Server:   +OK 2 2000
Client:   LIST
Server:   +OK 2 messages (2000 octets)
Server:   1 500
Server:   2 1500
Server:   .
```

```
Client:   RETR 1
Server:   +OK 500 octets
Server:   (…)
Server:   .
Client:   DELE 1
Server:   +OK message 1 deleted
Client:   RETR 2
Server:   +OK 1500 octets
Server:   (…)
Server:   .
Client:   DELE 2
Server:   +OK message 2 deleted
Client:   QUIT
Server:   +OK Bye
```

Le protocole IMAP

- IMAP = Internet Message Access Protocol (2003), port 143
- Fonctionnalité : consultation d'emails
  - Permet au même utilisateur de se connecter simultanément depuis plusieurs clients
  - Permet de télécharger des emails partiellement
  - Permet de gérer des répertoires
  - Permet de faire des recherches côté serveur

Le protocole IMAP – exemple

- Exemple :

```
Server:    * OK IMAP server ready
Client:    a001 login toto titi
Server:    a001 OK LOGIN completed
Client:    a002 select inbox
Server:    * 10 EXISTS
Server:    * 2 RECENT
Server:    * OK [UNSEEN 9] Message 9 is the
first unseen message
Server:    a002 OK SELECT completed
Client:    fetch 9 full
Server:    * 12 FETCH (FLAGS (\Seen)
       (…)
       BODY (" TEXT" "PLAIN" (….)))
```

```
Server:    a003 OK FETCH completed
Client:    a004 fetch 9 body[header]
Server:    * 12 FETCH (BODY[HEADER] (…)
       From: alice@isima.fr
       To: bob@isima.fr
       Subject: Test
       (…)
       )
       a004 OK FETCH completed
Client:    a005 store 9 +flags \deleted
Server:    a005 OK +FLAGS completed
Client:    a006 logout
Server:    * BYE
Server:    a006 OK LOGOUT completed
```