Architecture et Systèmes

Stefan Schwoon

Cours L3, 2016/17, ENS Cachan

Réseau

Applications:

- Communication entre utilisateurs (e-mail, web)
- Partager des données entre plusieurs machines (Network file system, NFS)
- Remote procedure call (RPC)
- Maintenance et backup automatisés
- ...

Modèle OSI

Décrit les tâches et les protocoles réquis pour la communiciation sur réseau

Recovre tous les niveaux (matériel, logiciel)

OSI = Open Systems Interconnection, modèle standardisé par l'ISO (Organisation internationale de normalisation)

Objectifs :

- établir des notions pour bien séparer les différents problèmes à résoudre
- permettre une définition modulaire des protocoles

Les couches ISO

L'OSI définit sept couches de communication :

physique / liaison / réseau / transport / session / présentation / application

Les couches "basses" fournissent des fonctionnalités simples.

Les couches "hautes" s'en servent pour accomplir des tâches plus complexes.

Une couche sur un ordinateur communique avec une couche du même niveau sur un autre ordinateur.

En principe, les données entre deux utilisateurs/applications traversent des instances de tous les couches des deux côtés.

Couches physique et liaison

Les deux premières couches s'occupent de la communication entre deux machines directement liées.

Physique : Standards pour transmettre des bits (voltage, vitesse, etc)

Liaison : Grouper des bits/octets en paquets, détection d'erreurs

Exemples: Ethernet, ISDN, DSL, . . .

Matériel: repeater, hub, bridge, switch

Adresse MAC : Identifiant (6 octets) d'un interface physique (p.ex., carte réseau)

Couche réseau

La couche Réseau gère la communication entre deux machines à travers d'un réseau (envoi de paquets individuels).

Chaque machine est identifié par une *adresse réseau*.

Le protocole décide par quels liens les données seront acheminées.

Exemple : IP (Internet protocol)

- Machines identifiés par une adresse IP (32 bits pour IPv4, 128 pour IPv6)
- Adresse hiérachiques : partie haute identifie un sous-réseau, partie basse une machine dans un réseau

Couche transport

La couche Transport établit une “connection” entre deux processus sur deux machines.

Découper les données en paquets individuels (qui pourront prendre de chemins différents).

Protocoles importants :

UDP : protocole simple et efficace, mais sans traitement d’erreurs en cas de perte de données

TCP : assure que tous les paquets seront bien reçus et dans le bon ordre

Couches “logiciel”

Les couches session / représentation / application sont des notions pour la construction des logiciels et se confondent souvent.

Session : Preserver une ‘connection logique’ à travers de plusieurs connections physiques (services d’authentification, cookies, . . .)

Représentation : comment représenter des données (caractères, valeurs numériques, . . .)

Applications : des protocoles comme SSH, HTTP, FTP, . . .

Comment réaliser une connection fiable

Sur certains couches (notamment 2 et 4) on doit réaliser une connection fiable sur des canaux non-fiables. Hypothèses :

Le canal de communication transfère des paquets individuels.

Le temps d'un transfert est variable (sans aucune borne).

Le transfert d'un paquet peut échouer sans notification.

Le canal n'est pas complètement bloqué (de temps en temps, un paquet arrivera).

Les paquets arrivent dans l'ordre d'envoi.

On suppose que l'intégrité des paquets est assurée (par des checksum).

Scénario

On étudie un protocole simple pour le scénario suivant :

> Alice produit des messages $m_0, m_1, \ldots$ et les envoie à Bob. On assure que Bob recevoit tous les messages.

Hypothèse additionnel :

> Bob ne gère qu'un message à la fois.
>
> Du coup, Alice doit attendre la confirmation de Bob après chaque message.

Solution simple : Alternating-bit protocol

Alice gèrent Bob tous les deux leur propre bit “séquence” s_A and s_B, initialement 0 et 1, respectivement.

Alice possède un compteur i, initialement 0.

Les messages d’Alice sont de la forme (m_i, s_A).

Les messages de Bob sont de la forme s_B.

Le protocole

La boucle d'Alice consiste de deux étapes :

1. Alice envoie (m_i, s_A) et répète ce message de temps en temps.

2a. Quand Alice reçoit un bit égal à s_A de la part de Bob, elle augmente i et change la valeur de s_A.

2b. Si Alice reçoit un bit de Bob qui n'est pas égal à la valeur de s_A, elle ignore ce message et continue comme avant.

La boucle de Bob:

1. Bob envoie s_B et répète ce message de temps en temps.

2a. Quand Bob reçoit $(m, \neg s_B)$, il stocke le message m et change la valeur de s_B.

2b. Si Bob reçoit (m, s_B), il ignore le message.

Remarques

Ce protocole construit un flux de communication fiable sur un canal non-fiable, sous les hypothèses évoqués.

On peut rendre le protocole plus efficace en utilisant un ensemble plus grand d'identifiants (plus qu'un bit, un mot entier ...) ; ne pas attendre la confirmation pour chaque message, adapter seulement en cas d'erreur.

Protection contre les arrivées hors ordre : étiquetter chaque paquet avec un horloge, éliminer des messages trop vieux.

Les ports

Dans les protocoles TCP et UDP une addresse consiste d'une *adresse machine* (32 ou 128 bits, selon la version d'IP) et un *port*.

Les ports permettent à une machine d'avoir plusieurs connections IP dans un même temps : le nombre de ports est de 2^{16}.

Les ports 0..1023 sont typiquement reservés pour certains protocoles, p.ex. 80 pour HTTP.

Les sockets

En POSIX, un socket est une structure que permet de communiquer sur un port IP.

Un descripteur de fichier peut être lié à un socket.

- Création d'un socket : appel `socket`(2).
- Lier un socket à un port : appels `bind`(2) ou `connect`(2).
- Envoi/reception de données : `read`, `write`

Modèle typique :

- Serveur : ouvre un port sur sa machine et attend une connection.
- Client : se connecte au port d'un serveur

Serveur

Typiquement, un serveur fait les étapes suivants (voir l'exemple) :

1. Créer un socket socket (avec `socket`).

2. Connecter le socket à un port IP (avec `bind`).

3. Configurer le socket en mode écoute (avec `listen`).

4. Attendre un client (avec `accept`).

Remarque: accept renvoie un descripteur qui sert uniquement à communiquer avec ce client. Du coup, on peut communiquer avec plusieurs clients sur un même socket.

Client

Un client typique :

1. Créer un socket TCP (comme dans le serveur).

2. Connecter le socket à un port du serveur (avec `connect`).

Après connect, le socket est lié à un port libre du client qui est en communication avec le bon port du serveur. Le socket peut être utilisé comme descripteur pour envoyer/recevoir des données.