

TP Réseaux : Interrogation d'un serveur DNS

L'objectif de ce TP est de manipuler le protocole UDP en C et en Java et d'interroger un serveur DNS (serveur qui traduit les adresses symboliques en adresses IP).

1 Le protocole UDP

On rappelle qu'UDP est un protocole de transport (couche 4 du modèle OSI) sans connexion qui fonctionne au dessus du protocole de réseau IP (couche 3 du modèle OSI). C'est un protocole simple à mettre en oeuvre, cependant il n'est pas fiable (perte de messages, messages non ordonnés, ...). Les messages qu'envoie UDP sont appelés datagrammes.

1.1 Recevoir un message UDP

Il s'agit ici de recevoir un message transmis à l'aide du protocole UDP. En Java, les classes importantes pour réaliser ce programme sont :

- La classe `DatagramPacket` et notamment 2 de ses constructeurs :
 - `DatagramPacket(byte[] buf, int length, InetAddress address, int port);` crée un datagramme de `length` octets contenus dans `buf`. Ces octets seront envoyés à l'adresse `address` sur le port `port`.
 - `DatagramPacket(byte[] buf, int length);` crée un datagramme pour recevoir des données, ces données seront stockées dans le buffer `buf` de taille `length`.
- La classe `InetAddress` pour construire une adresse internet.
- La classe `DatagramSocket` qui permet d'obtenir une socket pour échanger des datagrammes (notamment avec les méthodes `send` et `receive`).

Question 1 : *Ecrire un programme en Java qui récupère un datagramme UDP et qui affiche le message reçu sous forme d'une chaîne de caractères (on pourra utiliser le constructeur `String(byte[])`).*

1.2 Emettre un message UDP

Il s'agit ici de créer un message et de la transmettre à une autre machine en utilisant le protocole UDP. Pour définir une adresse réseau en C, on utilise la structure `sockaddr_in` définie dans `<netinet/inet.h>`

Cette structure est définie comme suit :

```
struct sockaddr_in
{
    short  sin_family; /* famille d'adresse = AF_INET */
    u_short sin_port; /* port UDP ou TCP à utiliser */
    struct in_addr sin_addr; /* 4 octets de l'adresse internet */
    char sin_zero[8]; /* doit valoir 0 */
};
```

Le champ `sin_addr` peut facilement être obtenu avec la fonction `inet_addr(const char *)`; qui à partir d'une adresse IP stockée dans une chaîne de caractères retourne la structure `in_addr` correspondante.

Pour pouvoir échanger des messages avec le protocole UDP, vous avez besoin des fonctions suivantes (consulter les pages du man pour obtenir les informations sur les différents paramètres) :

```
/* Création de la socket */
```

```

int socket(int domain, int type, int protocol);

/* Envoi d'un message */
int sendto(int s, const void * msg, size_t len, int flags,
           const struct sockaddr * to, socklen_t tolen);

/* Reception d'un message */
int recv(int s, void * buf, int len, unsigned int flags);

```

Question 2 : écrire un programme en C qui prend une chaîne de caractères en argument. Ce programme devra envoyer cette chaîne à l'aide du protocole UDP.

2 DNS : Domain Name Server

2.1 Introduction

DNS est un système hiérarchique distribué permettant de traduire des adresses symboliques en adresses IP. Les informations sont stockées sur des serveurs organisés en domaines hiérarchiques. Ainsi, un serveur peut déléguer une partie des noms du domaine à un serveur subalterne.

On dit d'un serveur de noms responsable de la mise à jour des correspondances nom / adresse IP des machines de son domaine qu'il est serveur d'autorité pour le domaine (*Authorative Serveur*).

Pour contacter un serveur DNS, les messages doivent être envoyés avec le protocole UDP sur le port 53.

2.2 Les messages du protocole DNS

Les messages (requête ou réponse) du protocole DNS sont définis dans le RFC1035. La structure d'un message est la suivante :

| | |
|-------------|--|
| En-tête | <i>spécifie le type du message (taille fixe : 12 octets)</i> |
| Question | <i>question posée au serveur de nom</i> |
| Réponse | <i>réponse à la requête</i> |
| Autorité | <i>information sur les serveurs d'autorité</i> |
| Additionnel | <i>informations complémentaires</i> |

2.2.1 Champs d'en-tête d'un datagramme UDP pour une résolution DNS

| | |
|-------------|------------|
| Identifiant | Paramètres |
| QDcount | Ancount |
| NScount | ARcount |

Chaque champ de l'en-tête est codé sur 16 bits.

- *identifiant* est un entier permettant d'identifier la requête.
- *paramètres* contient les champs suivant :
 - QR (1 bit) : indique si le message est une question (0) ou une réponse (1).
 - OPCODE (4 bits) : type de la requête (0000 pour une requête simple).
 - AA (1 bit) : le serveur qui a fourni la réponse a-t'il autorité sur le domaine ?
 - TC (1 bit) : indique si le message est tronqué.
 - RD (1 bit) : demande d'une requête récursive.
 - RA (1 bit) : indique que le serveur peut faire une demande récursive.
 - UNUSED, AD, CD (1 bit chacun) : non utilisés.

- RCODE (4 bits) : code de retour. 0 : OK, 1 : erreur sur le format de la requête, 2: problème du serveur, 3 : nom de domaine non trouvé (valide seulement si AA), 4 : requête non supportée, 5 : le serveur refuse de répondre (raisons de sécurité ou autres).
- *QDCOUNT* : nombre de questions.
- *ANCOUNT*, *NSCOUNT*, *ARCOUNT* : nombre d'entrées dans les champs "Réponse", "Autorité", "Additionnel".

2.2.2 Champ de requête DNS dans un datagramme UDP

Une question est représentée de la manière suivante :

- Label. Par exemple `www.lifl.fr` donne `0377 7777 046c 6966 6c02 6672`. Les points ne sont pas codés par contre on utilise des séparateurs (`02 03 04 ...`). Ainsi le premier octet `03` indique qu'il y a 3 octets avant le nouveau séparateur (i.e. `77 77 77` soit `www`), de même `04` indique qu'il y a 4 octets avant le nouveau séparateur (`6c 69 66 6c` soit `lifl`) et le dernier séparateur (`02`) indique qu'il y a 2 octets après : `66 72` soit `fr`.
- `00` pour indiquer la fin du nom.
- Type (16 bits) pour indiquer le type de la requête.
- Class (16 bits) indique le type du protocole.

Exemple de requête (trame émise en faisant `nslookup www.lifl.fr`):

```
08bb 0100 : identifiant 08bb - parametre 0100 (requête récursive)
0001 0000 : 1 question, 0 réponse
0000 0000 : 0 autorité, 0 infos complémentaire
0377 7777 : .w ww
046c 6966 : .l if
6c02 6672 : l. fr
00
00 01 : type (host address)
00 01 : class (internet)
```

Question 3 : écrire une fonction (en Java) qui permet de créer une requête DNS et envoyez là au serveur DNS du LIFL (adresse IP : 134.206.10.18).

2.2.3 Autres champs d'un paquet DNS

Les champs *Réponse*, *Autorité*, *Additionnel* sont tous représentés de la même manière :

- Nom (16 bits) : Pour éviter de recopier la totalité du nom, on utilise des offsets. Par exemple si ce champ vaut `C0 0C`, cela signifie qu'on a un offset (`C0`) de 12 (`0C`) octets. C'est-à-dire que le nom en clair se trouve au 12ème octet du message.
- Type (16 bits) : idem que pour le champ *Question*.
- Class (16 bits) : idem que pour le champ *Question*.
- TTL (32 bits) : durée de vie de l'entrée.
- RDLenght (16 bits) : nombre d'octets de la zone RRDData.
- RRDData (RDLenght octets) : réponse

Question 4 : toujours en Java, récupérer la trame de retour fournie par le serveur, l'afficher à l'écran, l'analyser et isoler l'adresse IP.

Question 5 : en Java, écrire une méthode statique qui prend une adresse symbolique sous la forme d'une chaîne de caractères (type *String*) et retourne une adresse IP sous la forme d'un entier (type *int*, 32 bits).

Question 6 : faire la même chose en C.