# COMPTE RENDU TRAVAUX PRATIQUES DE LANGAGE C

## TP 4: Communication réseaux - Protocole UDP

**Etudiant :** Assane Thiao

**Formation :** Télécommunications et Réseaux

**Niveau** : ING1

**Année :** 2024/2025

# 1. Objectifs

Ce TP permet d'apprendre à réaliser des communications réseau via UDP. L'objectif est d'écrire un client et un serveur UDP en utilisant des sockets pour envoyer et recevoir des messages.

# 2. Tâches

## 1. Client UDP :

- Créer un client UDP qui envoie un message au serveur local sur le port 8080 en utilisant `sendto()`.
- Modifier pour recevoir la réponse du serveur avec `recvfrom()`.
- Permettre de choisir un port d'envoi via la ligne de commande, avec un port par défaut (8080).

## 2. Serveur UDP :

- Créer un serveur UDP qui écoute sur le port 8080 et affiche les messages reçus.
- Répondre à chaque client avec un message de confirmation.
- Fermer le serveur si le message "SERVER:QUIT" est reçu.

## 3. Communication à distance :

- Envoyer un message broadcast sur le port 50555 pour découvrir un serveur distant.
- Envoyer un message de validation au serveur une fois son adresse et son port récupérés.

# 3. Code

## Client UDP :

```c
/*Assane Thiao 12/03/2025*/

#include "network.h"  /* Inclure le fichier d'en-tête personnalisé */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define SERVER_PORT 8080
#define SERVER_IP "127.0.0.1"  /* L'adresse IP du serveur (127.0.0.1 pour localhost) */
#define BUFFER_SIZE 1024

int main() {
    SOCKET sock;  /* Déclaration du socket */
    struct sockaddr_in server_addr;  /* Déclaration de la structure d'adresse du serveur */
    char buffer[BUFFER_SIZE];  /* Buffer pour stocker les données reçues */

    /* Initialisation du réseau */
    initNetwork();

    /* Création du socket */
    sock = socket(AF_INET, SOCK_DGRAM, 0);
    if (sock == INVALID_SOCKET) {
        /* Erreur lors de la création du socket */
        printf("Erreur lors de la création du socket\n");
        cleanNetwork();  /* Nettoyage du réseau avant de quitter */
        return EXIT_FAILURE;
    }

    /* Configuration de l'adresse du serveur */
    memset(&server_addr, 0, sizeof(server_addr));  /* Initialisation de la structure server_addr à zéro */
    server_addr.sin_family = AF_INET;  /* Spécifie la famille d'adresses (IPv4) */
    server_addr.sin_port = htons(SERVER_PORT);  /* Spécifie le port du serveur */
    server_addr.sin_addr.s_addr = inet_addr(SERVER_IP);  /* Spécifie l'adresse IP du serveur */

    /* Envoi du message au serveur */
    const char* message = "Hello from client";
```

```c
    /* Envoi du message au serveur */
    const char* message = "Hello from client";
    int send_len = sendto(sock, message, strlen(message), 0, (struct sockaddr*)&server_addr, sizeof(server_addr));
    if (send_len == SOCKET_ERROR) {
        /* Erreur lors de l'envoi du message */
        printf("Erreur lors de l'envoi du message: %d\n", WSAGetLastError());
        closesocket(sock);  /* Fermeture du socket */
        cleanNetwork();  /* Nettoyage du réseau */
        return EXIT_FAILURE;
    }

    printf("Message envoye au serveur: %s\n", message);  /* Affichage du message envoyé au serveur */

    /* Réception de la réponse du serveur */
    int recv_len = recvfrom(sock, buffer, sizeof(buffer) - 1, 0, NULL, NULL);
    if (recv_len == SOCKET_ERROR) {
        /* Erreur lors de la réception du message */
        printf("Erreur lors de la réception du message: %d\n", WSAGetLastError());
        closesocket(sock);  /* Fermeture du socket */
        cleanNetwork();  /* Nettoyage du réseau */
        return EXIT_FAILURE;
    }

    buffer[recv_len] = '\0';  /* Ajouter un caractère de fin de chaîne */
    printf("Reponse du serveur: %s\n", buffer);  /* Affichage de la réponse du serveur */

    /* Fermeture du socket */
    closesocket(sock);
    cleanNetwork();  /* Nettoyage du réseau avant de quitter */
    return 0;
}
```

## Serveur UDP :

```c
/*Assane Thiao 12/03/2025*/

#include "network.h"  /* Inclure le fichier d'en-tête personnalisé */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define PORT 8080
#define BUFFER_SIZE 1024

int main() {
    SOCKET sock;  /* Déclaration du socket */
    struct sockaddr_in server_addr, client_addr;  /* Structures pour l'adresse du serveur et du client */
    char buffer[BUFFER_SIZE];  /* Buffer pour stocker les données reçues */
    int client_addr_len = sizeof(client_addr);  /* Taille de l'adresse du client */

    /* Initialisation du réseau */
    initNetwork();

    /* Création du socket */
    sock = socket(AF_INET, SOCK_DGRAM, 0);
    if (sock == INVALID_SOCKET) {
        /* Erreur lors de la création du socket */
        printf("Erreur lors de la création du socket\n");
        cleanNetwork();  /* Nettoyage du réseau avant de quitter */
        return EXIT_FAILURE;
    }

    /* Configuration de l'adresse du serveur */
    memset(&server_addr, 0, sizeof(server_addr));  /* Initialisation de la structure server_addr à zéro */
    server_addr.sin_family = AF_INET;  /* Spécifie la famille d'adresses (IPv4) */
    server_addr.sin_port = htons(PORT);  /* Spécifie le port du serveur */
    server_addr.sin_addr.s_addr = INADDR_ANY;  /* Spécifie l'adresse IP du serveur (toutes les interfaces réseau) */

    /* Lier le socket à l'adresse du serveur */
    if (bind(sock, (struct sockaddr*)&server_addr, sizeof(server_addr)) == SOCKET_ERROR) {
```

```c
    /* Lier le socket à l'adresse du serveur */
    if (bind(sock, (struct sockaddr*)&server_addr, sizeof(server_addr)) == SOCKET_ERROR) {
        /* Erreur lors de la liaison du socket */
        printf("Erreur lors de la liaison du socket\n");
        closesocket(sock);  /* Fermeture du socket */
        cleanNetwork();  /* Nettoyage du réseau */
        return EXIT_FAILURE;
    }

    printf("Serveur en ecoute sur le port %d...\n", PORT);

    /* Boucle infinie pour rester en écoute */
    while (1) {
        /* Réception des données du client */
        int recv_len = recvfrom(sock, buffer, sizeof(buffer) - 1, 0, (struct sockaddr*)&client_addr, &client_addr_len);
        if (recv_len == SOCKET_ERROR) {
            /* Gestion des erreurs lors de la réception des données */
            int error_code = WSAGetLastError();
            if (error_code == WSAEINTR) {
                /* Si l'appel est interrompu, simplement recommencer */
                continue;
            }
            /* Affichage de l'erreur */
            printf("Erreur lors de la reception du message: %d\n", error_code);
            break;  /* Sortir de la boucle si l'erreur est fatale */
        }

        buffer[recv_len] = '\0';  /* Ajout d'un caractère de fin de chaîne */
        printf("Message recu du client: %s\n", buffer);

        /* Répondre au client */
        const char* response = "Message recu avec succes!";
        int send_len = sendto(sock, response, strlen(response), 0, (struct sockaddr*)&client_addr, client_addr_len);
        if (send_len == SOCKET_ERROR) {
```

```c
        if (send_len == SOCKET_ERROR) {
            /* Erreur lors de l'envoi du message */
            int error_code = WSAGetLastError();
            printf("Erreur lors de l'envoi du message: %d\n", error_code);
        }
        else {
            /* Affichage de la réponse envoyée */
            printf("Reponse envoyee au client: %s\n", response);
        }
    }

    /* Fermeture du socket */
    closesocket(sock);
    cleanNetwork();  /* Nettoyage du réseau avant de quitter */
    return 0;
}
```

# 4. Conclusion

Ce TP a permis de comprendre la communication via UDP, de créer un client et un serveur pour échanger des messages, et de gérer des erreurs. Nous avons également appris à configurer des ports et à établir des connexions à distance.