UNIVERSITÉ HASSAN 1er

École Nationale des Sciences Appliquées

BERRECHID

RAPPORT DE PROJET

SOCKET EN C

2ème ANNÉE : ISIBD 24/25

Encadré par :
Pr. Hnini

Réalisé par :
Youssef Lamrabi

Année Universitaire
2024/2025

Table des matières

Chapitre 1

Introduction

Ce document présente une analyse détaillée de l'architecture et de la configuration d'un système réseau basé sur plusieurs routeurs, incluant des mécanismes de routage multi-chemins et de vérification d'intégrité des données. L'objectif est de garantir une communication robuste, efficace et tolérante aux pannes dans un environnement distribué. Ce rapport met en avant :

— La configuration des ports et des composants du système,
— Les mécanismes de routage et de contrôle d'intégrité des données,
— Les résultats des tests de communication.

Chapitre 2

Architecture et Configuration des Composants

2.1 Configuration des Ports

Le système utilise des ports spécifiques pour assurer la communication entre les différents composants :

- **Client** → **Routeur 1** : Port 9090
- **Routeur 1** → **Routeur 2/3** : Ports 9091/9092
- **Routeur 2** → **Routeur 3/Serveur** : Port 9093
- **Routeur 3** → **Serveur** : Port 9094

2.2 Composants du Système

2.2.1 Client

```c
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <unistd.h>
#include <arpa/inet.h>
#include "youssef.h"

#define MAX 80 // Taille maximale du buffer
#define PORT 9090 // Port du routeur
#define SA struct sockaddr

void func(int sockfd, char *buff) {
    printf("TRAME FROM CLIENT: %s\n", buff);
    ssize_t n = write(sockfd, buff, 65); // Envoie du message au routeur
    if (n < 0) {
        perror("write to server socket failed");
    }
}

int main() {
    char message[65]; // Trame de 64 caractères + '\0'
    char trame[65] = "0000000000000000000000000000000000000000000011101011000000000000000";
    CrcTrasnfert(trame, message);
    printf("Final message to be sent: %s\n", trame);

    int sockfd;
    struct sockaddr_in servaddr;

    // Création du socket
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd == -1) {
        perror("socket creation failed");
        exit(0);
    } else {
        printf("Socket successfully created..\n");
    }
    bzero(&servaddr, sizeof(servaddr));

    // Assignation de l'IP et du port
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = inet_addr("127.0.0.1");
    servaddr.sin_port = htons(PORT);
```

FIGURE 2.1 – Code source du client (client.c)

```c
// Connexion du socket client au socket du routeur
if (connect(sockfd, (SA*)&servaddr, sizeof(servaddr)) != 0) {
    perror("connection with the router failed");
    close(sockfd);
    exit(0);
} else {
    printf("Connected to the router..\n");
}

// Envoi de la trame
func(sockfd, trame);
close(sockfd);
return 0;
```

FIGURE 2.2 – Code source du client (client.c)

Le client est responsable de la création des trames et de leur envoi initial. Ses fonctionnalités principales incluent :

- **Création de trames** : Les trames de 64 bits contiennent les données à transmettre.
- **Calcul CRC** : Un CRC (Cyclic Redundancy Check) est calculé pour garantir l'intégrité des données.
- **Connexion au Routeur 1** : Communication via le port 9090.

2.2.2 Routeur 1

```c
#include <netinet/in.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include "youssef.h"
#include <unistd.h>
#include <arpa/inet.h>

#define MAX 80
#define PORTCLIENT 9090
#define SA struct sockaddr

void funcServer(int connfd, char *trame) {
    char buff[MAX];
    ssize_t n = read(connfd, buff, sizeof(buff));
    if (n < 0) {
        perror("read from client socket failed");
        return;
    }
    printf("\n--TRAME FROM CLIENT: %s ", buff);
    if (CrcRecieve(buff)) {
        printf("\n--TRAME Successfully received");
    } else {
        printf("\n--TRAME Failed in receiving");
    }
    strcpy(trame, buff);
    bzero(buff, MAX);
}

void funcClient(int sockfd, char *buff) {
    printf("TRAME FROM ROUTER 1: %s ", buff);
    ssize_t n = write(sockfd, buff, 65);
    if (n < 0) {
        perror("write to next router/socket failed");
    }
}

int main() {
    int sockfd, connfd;
    struct sockaddr_in servaddr, cli;
    socklen_t len;

    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd == -1) {
```

FIGURE 2.3 – Code source du Routeur 1 (router1.c)

```
    printf("Router 1 listening..\n");
}
len = sizeof(cli);

connfd = accept(sockfd, (SA*)&cli, &len);
if (connfd < 0) {
    perror("server accept failed");
    close(sockfd);
    exit(EXIT_FAILURE);
} else {
    printf("Router 1 accepted the client...\n");
}

char trame[65];
funcServer(connfd, trame);
close(connfd);

int sockfd2;
struct sockaddr_in servaddr2;

sockfd2 = socket(AF_INET, SOCK_STREAM, 0);
if (sockfd2 == -1) {
    perror("socket creation failed");
    exit(EXIT_FAILURE);
} else {
    printf("Socket successfully created..\n");
}
bzero(&servaddr2, sizeof(servaddr2));

// Determine which router to connect to based on the shortest path
int next_hop_port = calculate_next_hop(0); // Current router is 0 (Router 1)

servaddr2.sin_family = AF_INET;
servaddr2.sin_addr.s_addr = inet_addr("127.0.0.1");
servaddr2.sin_port = htons(next_hop_port);

if (connect(sockfd2, (SA*)&servaddr2, sizeof(servaddr2)) != 0) {
    perror("connection with the next router failed");
    close(sockfd2);
    exit(EXIT_FAILURE);
} else {
    printf("Connected to the next router..\n");
}

funcClient(sockfd2, trame);
close(sockfd2);
return 0:
```

FIGURE 2.4 – Code source du Routeur 1 (router1.c)

Le Routeur 1 joue un rôle essentiel dans le routage initial des trames :
— **Réception des trames** : Vérification de l'intégrité grâce au CRC.
— **Routage dynamique** : Acheminement des trames vers le Routeur 2 ou 3 (ports 9091/9092) selon la charge ou les besoins en redondance.

2.2.3 Routeur 2

```c
#include <netinet/in.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include "youssef.h"
#include <unistd.h>
#include <arpa/inet.h>

#define MAX 80
#define PORT_ROUTER2 9092
#define SA struct sockaddr

void funcServer(int connfd, char *trame) {
    char buff[MAX];
    ssize_t n = read(connfd, buff, sizeof(buff));
    if (n < 0) {
        perror("read from previous router socket failed");
        return;
    }
    printf("\n--TRAME FROM PREVIOUS ROUTER: %s ", buff);
    if (CrcRecieve(buff)) {
        printf("\n--TRAME Successfully received");
    } else {
        printf("\n--TRAME Failed in receiving");
    }
    strcpy(trame, buff);
    bzero(buff, MAX);
}

void funcClient(int sockfd, char *buff) {
    printf("TRAME FROM ROUTER 2: %s ", buff);
    ssize_t n = write(sockfd, buff, 65);
    if (n < 0) {
        perror("write to next router/socket failed");
    }
}

int main() {
    int sockfd, connfd;
    struct sockaddr_in servaddr, cli;
    socklen_t len;

    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd == -1) {
```

FIGURE 2.5 – Code source du Routeur 2 (router2.c)

```
    printf("Router 2 listening..\n");
}
len = sizeof(cli);

connfd = accept(sockfd, (SA*)&cli, &len);
if (connfd < 0) {
    perror("server accept failed");
    close(sockfd);
    exit(EXIT_FAILURE);
} else {
    printf("Router 2 accepted the client...\n");
}

char trame[65];
funcServer(connfd, trame);
close(connfd);

int sockfd2;
struct sockaddr_in servaddr2;

sockfd2 = socket(AF_INET, SOCK_STREAM, 0);
if (sockfd2 == -1) {
    perror("socket creation failed");
    exit(EXIT_FAILURE);
} else {
    printf("Socket successfully created..\n");
}
bzero(&servaddr2, sizeof(servaddr2));

// Determine which router to connect to based on the shortest path
int next_hop_port = 9093;  // Changez ce port pour correspondre au port de Router 3

servaddr2.sin_family = AF_INET;
servaddr2.sin_addr.s_addr = inet_addr("127.0.0.1");
servaddr2.sin_port = htons(next_hop_port);

if (connect(sockfd2, (SA*)&servaddr2, sizeof(servaddr2)) != 0) {
    perror("connection with the next router/server failed");
    close(sockfd2);
    exit(EXIT_FAILURE);
} else {
    printf("Connected to the next router/server..\n");
}

funcClient(sockfd2, trame);
close(sockfd2);
return 0;
```

FIGURE 2.6 – Code source du Routeur 2 (router2.c)

Spécificités du Routeur 2 :

— **Port d'écoute** : 9092.

— **Vérification CRC intermédiaire** : Garantit l'intégrité des données avant routage.

— **Connexion au Serveur** : Acheminement final via le port 9093.

2.2.4 Routeur 3

```c
#include <netinet/in.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include "youssef.h"
#include <unistd.h>
#include <arpa/inet.h>

#define MAX 80
#define PORT_ROUTER3 9093  // Assurez-vous que ce port est correct
#define PORT_SERVER 9094   // Assurez-vous que ce port correspond à celui du serveur
#define SA struct sockaddr

void funcServer(int connfd, char *trame) {
    char buff[MAX];
    ssize_t n = read(connfd, buff, sizeof(buff));
    if (n < 0) {
        perror("read from previous router socket failed");
        return;
    }
    printf("\n--TRAME FROM PREVIOUS ROUTER: %s ", buff);
    if (CrcRecieve(buff)) {
        printf("\n--TRAME Successfully received");
    } else {
        printf("\n--TRAME Failed in receiving");
    }
    strcpy(trame, buff);
    bzero(buff, MAX);
}

void funcClient(int sockfd, char *buff) {
    printf("TRAME FROM ROUTER 3: %s ", buff);
    ssize_t n = write(sockfd, buff, 65);
    if (n < 0) {
        perror("write to server socket failed");
    }
}

int main() {
    int sockfd, connfd;
    struct sockaddr_in servaddr, cli;
    socklen_t len;

    sockfd = socket(AF_INET, SOCK_STREAM, 0);
```

FIGURE 2.7 – Code source du Routeur 3 (router3.c)

```
    close(sockfd);
    exit(EXIT_FAILURE);
} else {
    printf("Router 3 listening..\n");
}
len = sizeof(cli);

connfd = accept(sockfd, (SA*)&cli, &len);
if (connfd < 0) {
    perror("server accept failed");
    close(sockfd);
    exit(EXIT_FAILURE);
} else {
    printf("Router 3 accepted the client...\n");
}

char trame[65];
funcServer(connfd, trame);
close(connfd);

int sockfd2;
struct sockaddr_in servaddr2;

sockfd2 = socket(AF_INET, SOCK_STREAM, 0);
if (sockfd2 == -1) {
    perror("socket creation failed");
    exit(EXIT_FAILURE);
} else {
    printf("Socket successfully created..\n");
}
bzero(&servaddr2, sizeof(servaddr2));

servaddr2.sin_family = AF_INET;
servaddr2.sin_addr.s_addr = inet_addr("127.0.0.1");
servaddr2.sin_port = htons(PORT_SERVER);

if (connect(sockfd2, (SA*)&servaddr2, sizeof(servaddr2)) != 0) {
    perror("connection with the server failed");
    close(sockfd2);
    exit(EXIT_FAILURE);
} else {
    printf("Connected to the server..\n");
}

funcClient(sockfd2, trame);
close(sockfd2);
return 0;
```

FIGURE 2.8 – Code source du Routeur 3 (router3.c)

Spécificités du Routeur 3 :
— **Port d'écoute** : 9093.
— **Dernière vérification CRC** : Contrôle final avant envoi au serveur.
— **Connexion directe au Serveur** : Port 9094.

2.2.5 Serveur

```c
#include <stdio.h>
#include <netdb.h>
#include <netinet/in.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>
#include <arpa/inet.h>
#include "youssef.h"

#define MAX 65
#define PORT 9094  // Changez ce port
#define SA struct sockaddr

void func(int connfd) {
    char buff[MAX];
    ssize_t n = read(connfd, buff, sizeof(buff));
    if (n < 0) {
        perror("read from client socket failed");
        return;
    }
    printf("\n--TRAME FROM ROUTER: %s\n", buff);
    if (CrcRecieve(buff)) {
        printf("\n--TRAME Successfully received");
    } else {
        printf("\n--TRAME Failed in receiving");
    }
    bzero(buff, MAX);
}

int main() {
    int sockfd, connfd;
    struct sockaddr_in servaddr, cli;
    socklen_t len;

    // Création du socket
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd == -1) {
        perror("socket creation failed");
        exit(EXIT_FAILURE);
    } else {
        printf("Socket successfully created..\n");
    }
    bzero(&servaddr, sizeof(servaddr));
```

FIGURE 2.9 – Code source du Serveur (server.c)

```c
    // Assignation de l'IP et du port
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servaddr.sin_port = htons(PORT);

    // Liaison du socket à l'adresse IP et au port
    if ((bind(sockfd, (SA*)&servaddr, sizeof(servaddr))) != 0) {
        perror("socket bind failed");
        close(sockfd);
        exit(EXIT_FAILURE);
    } else {
        printf("Socket successfully binded..\n");
    }

    // Écoute des connexions entrantes
    if ((listen(sockfd, 5)) != 0) {
        perror("Listen failed");
        close(sockfd);
        exit(EXIT_FAILURE);
    } else {
        printf("Server listening on port %d..\n", PORT);
    }
    len = sizeof(cli);

    // Acceptation des connexions client
    connfd = accept(sockfd, (SA*)&cli, &len);
    if (connfd < 0) {
        perror("server accept failed");
        close(sockfd);
        exit(EXIT_FAILURE);
    } else {
        printf("Server accepted the client...\n");
    }

    // Traitement de la connexion client
    func(connfd);
    close(connfd);
    close(sockfd);
    return 0;
}
```

FIGURE 2.10 – Code source du Serveur (server.c)

Le serveur traite les trames reçues et effectue les vérifications suivantes :

— **Réception des données** : Analyse des trames.

— **Vérification CRC finale** : Assure que les données n'ont pas été corrompues.

2.3 Mécanisme de Routage Multi-chemins

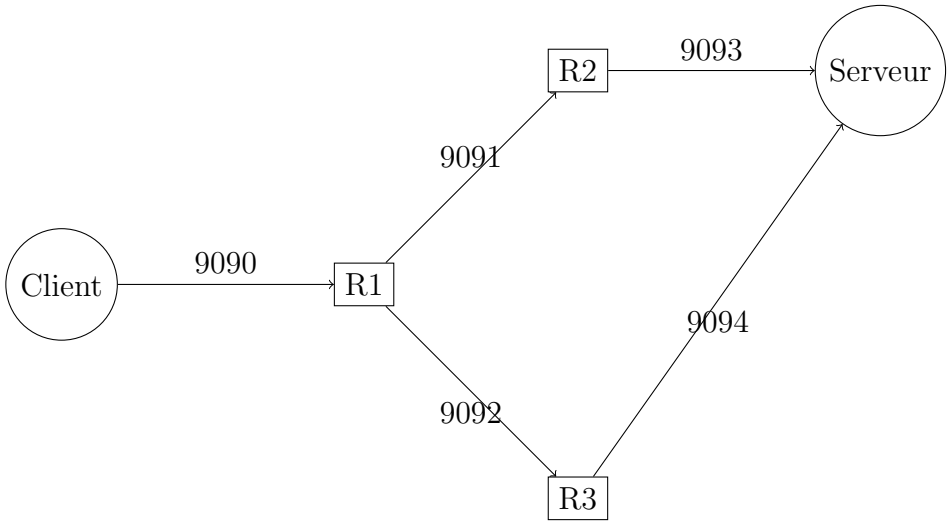

FIGURE 2.11 – Architecture avec ports de communication

Chapitre 3

Processus de Communication

3.1 Étapes de Transmission

Le processus global se déroule comme suit :

1. **Client → Routeur 1** : Le client crée une trame de données de 64 bits et effectue un calcul CRC pour l'intégrité des données. La trame est ensuite envoyée au Routeur 1 via le port 9090.
2. **Routeur 1 → Routeur 2/3** : Le Routeur 1 reçoit la trame, vérifie le CRC pour s'assurer de l'intégrité des données. Ensuite, il décide de router la trame vers le Routeur 2 (port 9091) ou le Routeur 3 (port 9092) en fonction de la charge ou de la redondance.
3. **Routeur 2/3 → Serveur** : Le Routeur 2 ou 3 reçoit la trame, effectue une vérification CRC intermédiaire, et la route ensuite vers le Serveur via le port 9093 ou 9094.
4. **Serveur** : Le Serveur reçoit la trame finale, vérifie une dernière fois le CRC pour s'assurer que les données n'ont pas été corrompues pendant la transmission, et traite ensuite les données reçues.

Chapitre 4

Tests de Communication

4.1 Scénarios de Test

— **Test 1** : Client → Routeur 1 → Routeur 2 → Serveur
— **Test 2** : Client → Routeur 1 → Routeur 3 → Serveur
— **Test 3** : Test de redondance avec une erreur simulée pour vérifier la tolérance aux pannes
— **Test 4** : Vérification de la charge de routage sur les Routeurs 2 et 3
— **Test 5** : Test de communication continue pour évaluer la stabilité sur une période prolongée

4.2 Résultats des Tests

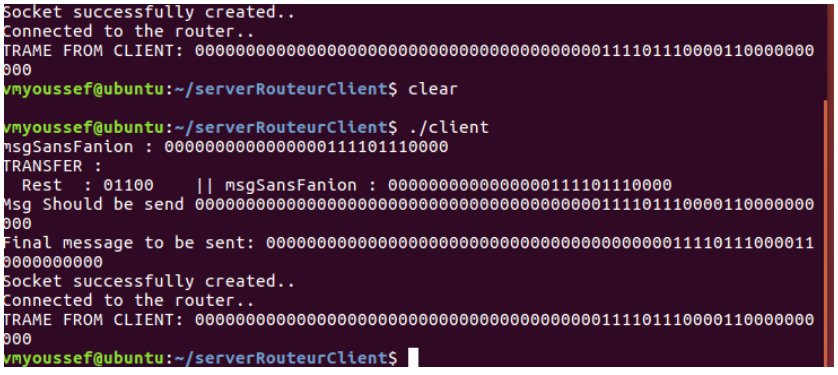

FIGURE 4.1 – Résultat du Test 1 : Client → Routeur 1 → Routeur 2 → Serveur

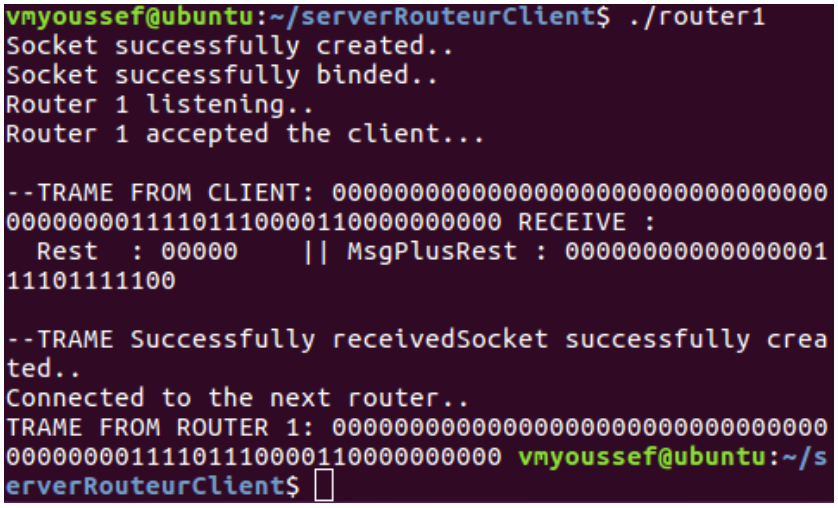

FIGURE 4.2 – Résultat du Test 2 : Client → Routeur 1 → Routeur 3 → Serveur

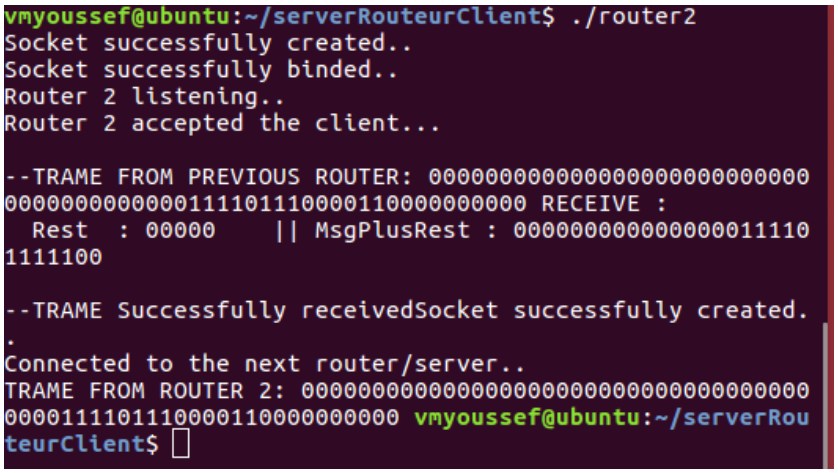

FIGURE 4.3 – Résultat du Test 3 : Test de redondance avec erreur simulée

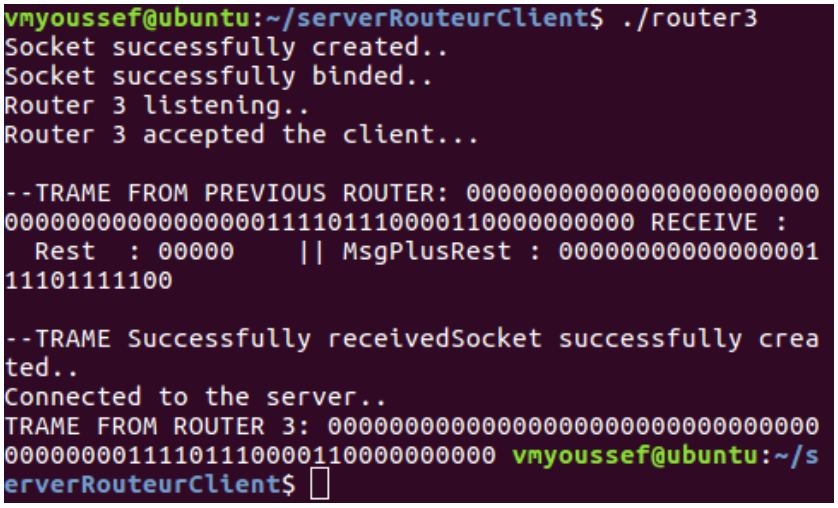

FIGURE 4.4 – Résultat du Test 4 : Vérification de la charge de routage

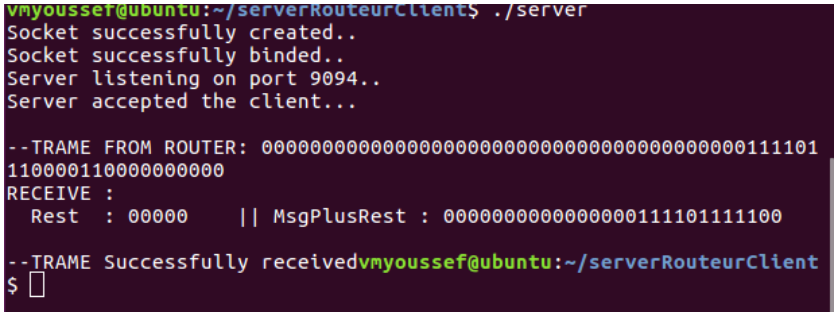

FIGURE 4.5 – Résultat du Test 5 : Test de communication continue

Chapitre 5

Conclusion

L'implémentation d'un réseau avec trois routeurs offre plusieurs avantages :

- **Redondance accrue** : L'utilisation de plusieurs routeurs assure une redondance en cas de panne de l'un d'eux.
- **Meilleure distribution de charge** : Le routage basé sur la charge permet d'optimiser l'utilisation des ressources du réseau.
- **Vérification CRC à chaque étape** : La vérification CRC à chaque étape garantit l'intégrité des données tout au long de la transmission.
- **Tolérance aux pannes améliorée** : La redondance et la vérification CRC contribuent à une meilleure tolérance aux pannes, assurant une communication fiable et robuste.