

2021/2022
LICENCE 3

RAPPORT DE ALGORITHMES DES RESEAUX

HUSEYIN ESAT
YILDIZTEPE
21811660

STEFAN
UNGUREANU
21801182

Table des matières

- Description.....4
 - 3 way handshake.....4
 - Stop and Wait.....5
 - Go-Back-N.....7
- Données utilisées.....10
- Structure du projet.....11
- Exécution du projet.....11

Description

Pour mettre en œuvre l'application qui génère du trafic UDP, nous avons implanté les mécanismes que nous allons décrire:

3 way handshake

Pour établir la connexion entre les deux entités que nous allons faire communiquer (c'est-à-dire *le client* et *le serveur* à travers du *perturbateur*), nous allons d'abord utiliser la **méthode 3 way handshake**.

Premièrement, le port du client va établir les paramètres d'un premier paquet qui contient le type de paquet SYN et va le renvoyer au serveur, en attendant de sa part un envoi du type SYN-ACK comme réponse que le serveur souhaite créer la connexion. Après, le client va renvoyer au serveur un paquet contenant le type ACK pour mettre en œuvre la connexion. En ce qui regarde la partie de l'implantation, nous avons procédé ainsi:

```
//Connection establishing
packet_send.id=1;
packet_send.type=1;
packet_send.seqNum=521;
packet_send.ecn=0;

printf("[+] SYN Packet Sent\n");
sendto(sockfd,&packet_send,sizeof(Packet), 0, (struct sockaddr *)&destAddr,sizeof(destAddr));

addr_size = sizeof(originAddr);
int f_rcv_size= recvfrom(sockfd,&packet_rcv, sizeof(Packet), 0,(struct sockaddr *)&originAddr,&addr_size);
if(f_rcv_size >0 && packet_rcv.type==17 && packet_rcv.acq==packet_send.seqNum+1 ){
    printf("[+] SYN-ACK Packet Received.\n");
    int ack_recieved=packet_rcv.acq;
    packet_send.id++;
    packet_send.type=16;
    packet_send.acq=packet_rcv.seqNum+1;
    packet_send.seqNum=ack_recieved;
    printf("[+] ACK Packet Sent\n");
    sendto(sockfd,&packet_send,sizeof(Packet), 0, (struct sockaddr *)&destAddr,sizeof(destAddr));
    printf("[+] Connection Established.\n");
} else {
    printf("[-] SYN-ACKPacket not recieved\n");
}
```

Le serveur de son côté, attend le premier paquet contenant le type SYN, renvoie au client le paquet contenant SYN et son acquittement ACK et à la fin il attend pour recevoir l'ACK du client pour que la connexion soit créée:

```
55 /* Établir la connexion */
56
57 addr_size = sizeof(serverAddr);
58 int f_rcv_size= recvfrom(sockfd,&packet_rcv, sizeof(Packet), 0,(struct sockaddr *)&serverAddr,&addr_size);
59 if(f_rcv_size >0 && packet_rcv.type==1 ){
60     printf("[+] SYN Packet Recieved\n");
61
62 }else{
63     printf("[-] SYN Packet not recieved\n");
64 }
65
66
67
68 int ack_rcv=1;
69 if(ack_rcv=1){
70
71     packet_send.id=2;
72     packet_send.type=17;
73     packet_send.acq=packet_rcv.seqNum+1;
74     packet_send.seqNum=2000;
75
76     sendto(sockfd,&packet_send,sizeof(Packet), 0, (struct sockaddr *)&mediumAddr,sizeof(mediumAddr));
77     printf("[+] SYN-ACK Packet Sent.\n");
78
79     int f_rcv_size= recvfrom(sockfd,&packet_rcv, sizeof(Packet), 0,(struct sockaddr *)&serverAddr,&addr_size);
80     if(f_rcv_size >0 && packet_rcv.type==16 && packet_rcv.acq==packet_send.seqNum+1){
81         printf("[+] ACK Packet Recieved\n");
82         ack_rcv=1;
83         printf("Connection Established\n");
84     }else{
85         printf("[-] ACK Packet not recieved\n");
86         ack_rcv=0;
87     }
88 }
```

Après avoir établi la connexion et créé les sockets pour la connexion et la communication, nous avons implanté deux mécanismes d'envoyer les paquets de données.

Stop and Wait

Le premier mécanismes que nous avons mis en œuvre est la méthode Stop and Wait.

D'abord nous avons défini un intervalle du temps de 2 secondes pour l'envoi d'un paquet du client au serveur.

```
102     tv.tv_sec = 2;
103     tv.tv_usec = 0;
104     setsockopt(sockfd, SOL_SOCKET, SO_RCVTIMEO, (char*)&tv, sizeof(struct timeval));
105     packet_send.fenetre=1;
106     packet_send.id=1;
107     packet_send.seqNum=0;
```

Après on démarre la communication, en utilisant une variable *dataFinish* pour créer une boucle infini:

```
109     int dataFinish=1;
110     while(dataFinish==1)
111     {
112         if(ack_rcv==1) {
113
114             packet_send.type=16;
115             packet_send.ecn=1;
116
117             printf("Enter Data: ");
118             scanf("%s",buffer);
119             strcpy(packet_send.data,buffer);
120
121             if((strcmp(buffer,"exit"))==0){
122                 Packet packet_fin;
123                 packet_fin.type=2;
124                 packet_fin.seqNum=92;
125                 printf("FIN packet sent\n");
126             }
```

Ensuite, nous commençons à envoyer les paquets, en utilisant les sockets et la fonction associée pour l'envoi du paquet au serveur, en marquant l'étape de la communication avec le champs *type* de la structure du paquet:

```
126
127     int closeConnection=0;
128     int closeFinACKLen;
129     while(closeConnection==0){
130         if (sendto(sockfd,&packet_fin,sizeof(Packet), 0, (struct sockaddr *)&destAddr,sizeof(destAddr)) != sizeof(Packet))
131             printf("Erreur de l'envoi\n");
132         int packetFinSeq=packet_fin.seqNum;
133         printf("PacketFinSeqNum: %d\n",packetFinSeq);
134         while(1){
135             printf("Waiting for FIN-ACK\n");
136             int originAddrLen=sizeof(originAddr);
137             if(closeFinACKLen = recvfrom(sockfd, &packet_fin, sizeof(Packet), 0,
138                 (struct sockaddr *) &originAddr, &originAddrLen) == -1)
139                 printf("[.] Error");
140             printf("Packet fin: acq=%d seqnum=%d\n",packet_fin.acq,packet_fin.seqNum);
141             if(packet_fin.acq==packetFinSeq+1 && packet_fin.type==18){
142                 packet_fin.acq=packet_fin.seqNum+1;
143                 if (sendto(sockfd,&packet_fin,sizeof(Packet), 0, (struct sockaddr *)&destAddr,sizeof(destAddr)) != sizeof(Packet))
144                     printf("Erreur de l'envoi\n");
145                 closeConnection=1;
146                 printf("Disconnecting...\n");
147                 dataFinish=0;
148                 return(EXIT_SUCCESS);
149             }
150         }
151     }
152 }
```

De sa part, le serveur reçoit les paquets de données et prépare le paquet contenant l'acquittement pour informer le client qu'il a bien reçu le paquet de données.

```
101 while(dataFinish){
102     int addr_size = sizeof(mediumAddr);
103     int f_rcv_size= recvfrom(sockfd,&packet_rcv, sizeof(Packet), 0,(struct sockaddr *)&mediumAddr,&addr_size);
104     if (f_rcv_size > 0) {
105         if (packet_rcv.type == 16) {
106             printf("[+] ACK Recieved\n ACK NO:%d\n Seq NO:%d\n",packet_rcv.acq,packet_rcv.seqNum);
107             printf("Message recieved:%s\n",packet_rcv.data);
108             ack_rcv=1;
109         } if (packet_rcv.type == 2) {
110             packet_send.acq = (packet_rcv.seqNum + 1);
111             packet_send.type = 18;
112             packet_send.seqNum = 97;
113             if (sendto(sockfd, &packet_send, sizeof(Packet), 0,
114                 (struct sockaddr *) &mediumAddr, sizeof(mediumAddr)) != sizeof(Packet))
115                 printf("Error\n");
116             int closeFinACKLen;
117             int serverAddrLen = sizeof(serverAddr);
118             if(closeFinACKLen = recvfrom(sockfd, &packet_rcv, sizeof(Packet), 0,
119                 (struct sockaddr *) &serverAddr, &serverAddrLen) == -1)
120                 printf("[-] Error\n");
121             if (packet_rcv.acq==packet_send.seqNum+1){
122                 printf("All messages has been received\n");
123                 printf("Disconnecting...\n");
124                 sleep(2);
125                 dataFinish = 0;
126             }
127     }
```

À la fin, le client va conclure le renvoi d'un paquet en recevant les paquets contenant les acquittement de la part du serveur:

```
159     int addr_size = sizeof(destAddr);
160     int f_rcv_size= recvfrom(sockfd,&packet_rcv, sizeof(Packet), 0,(struct sockaddr *)&destAddr,&addr_size);
161     if (f_rcv_size > 0 ) {
162         printf("[+] ACK Recieved %d from %d\n",packet_rcv.acq,packet_rcv.id);
163         packet_send.id=packet_rcv.id;
164         packet_send.seqNum=(packet_rcv.seqNum+1)%2;
165         packet_send.id++;
166         ack_rcv=1;
167     } else {
168         printf("[-] ACK not recieved\n");
169         printf("Packet %d Resending...\n",packet_send.id);
170         ack_rcv=0;
171     }
172
173     close(sockfd);
174
175 }
```

Nous allons répéter ce démarche jusqu'au moment où tous les paquets ont été envoyé et après la communication est terminés, en fermant le sockets qui crée l'environnement de communication.

Go-Back-N

En ce qui concerne la méthode d'envoi de paquets **Go-Back-N**, les choses se compliquent un peu, puisque nous devons gérer la taille de toute la fenêtre contenant les paquets que nous allons envoyer du client au serveur, ainsi que la fenêtre glissante défini par la taille du segments que nous voulons envoyer à chaque fois avant de recevoir l'ensemble d'acquittements de la part du serveur.

Pour commencer, nous avons utilisé deux variables *chunkSize* (représentant la taille de chaque segment du paquet que nous allons envoyer) et *windowSize* (la taille de toute la fenêtre des paquets que nous allons envoyer).

D'abord nous allons commencer par diviser le contenu du paquet que nous voulons envoyer en fonction des deux variables décrites dans le paragraphe précédent pour établir comment nous allons envoyer chaque portion de données au serveur.

```
while (probACK && (indexBuffer < DATA_COUNT)) {
    char seqNum = 0;
    int tailleBuffer = strlen(buffer[indexBuffer]);
    printf("%d\n", tailleBuffer);
    int numSegments = tailleBuffer / chunkSize;
    if (strlen(buffer[indexBuffer]) % chunkSize > 0)
        numSegments++;

    while (seqNum < numSegments) {

        char seg_data[chunkSize];
        strncpy(seg_data, (buffer[indexBuffer] + (idValue%10) * chunkSize), chunkSize);

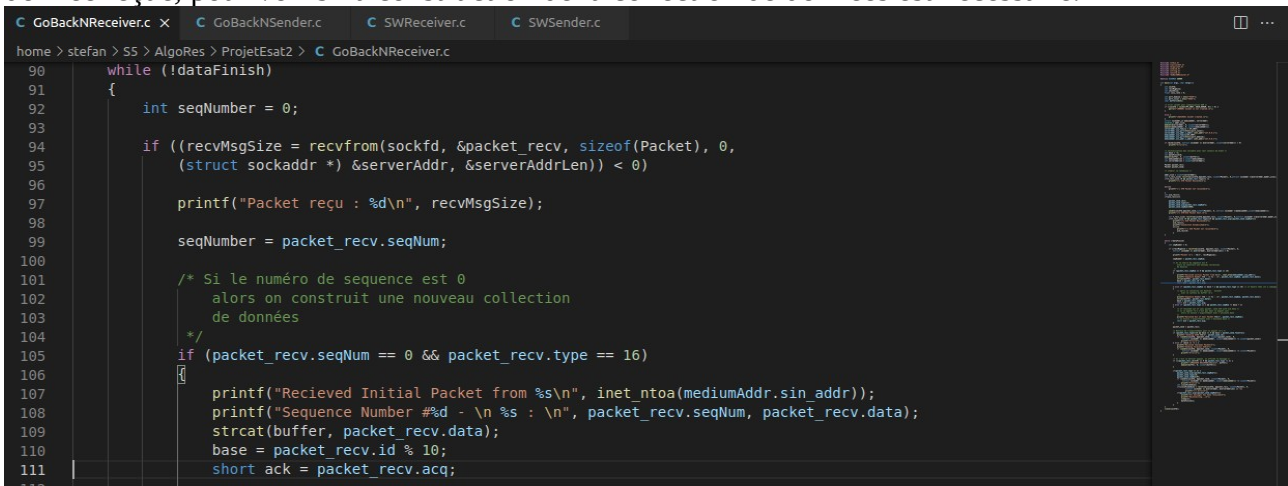
        packet_send.id = idValue;
        packet_send.seqNum = seqNum;
        packet_send.type = 16;
        packet_send.acq = idValue;
        packet_send.fenetre = windowSize;
        packet_send.length = chunkSize;
        strcpy(packet_send.data, seg_data);
        printf("Sending Packet : %d - %s - ACK : %d - Data: %s\n", idValue, packet_send.data, packet_send.acq, packet_ser
        memset(seg_data, 0, sizeof(seg_data));
```

probACK est une variable qui nous permet de faire marcher la boucle pour envoyer les paquets et la deuxième condition assure au programme que nous allons arrêter l'envoi au moment où nous allons renvoyer tous les segments.

Ensuite nous allons commencer à renvoyer ces segments au serveur:

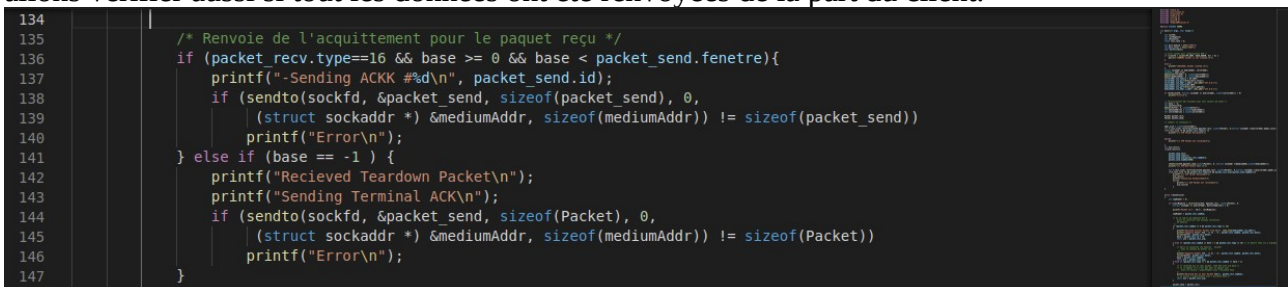
```
133
134     if (sendto(sockfd, &packet_send, sizeof(Packet), 0, (struct sockaddr *)&destAddr, sizeof(destAddr)) != sizeof(packet_send))
135         printf("Erreur de l'envoi\n");
136
137     seqNum++;
138     idValue++;
139
140
```


Le serveur de son côté va premièrement recevoir les premiers segments équivalents à la taille établie au moment de la création de la fenêtre et va d'abord vérifier si c'est la première partie de donnée reçue, pour voir si la construction de la collection de données est nécessaire.



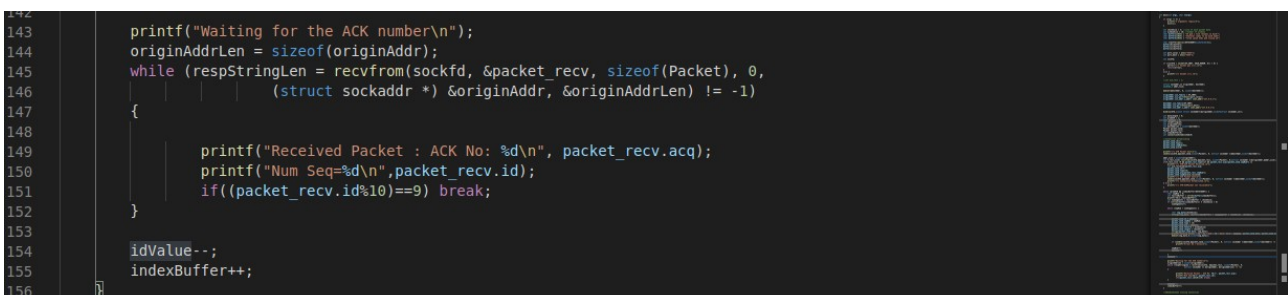
```
home > stefan > SS > AlgoRes > ProjetEsat2 > C GoBackNReceiver.c
90 while (!dataFinish)
91 {
92     int seqNumber = 0;
93
94     if ((recvMsgSize = recvfrom(sockfd, &packet_recv, sizeof(Packet), 0,
95         (struct sockaddr *) &serverAddr, &serverAddrLen)) < 0)
96
97         printf("Packet reçu : %d\n", recvMsgSize);
98
99         seqNumber = packet_recv.seqNum;
100
101         /* Si le numéro de sequence est 0
102            alors on construit une nouveau collection
103            de données
104         */
105         if (packet_recv.seqNum == 0 && packet_recv.type == 16)
106         {
107             printf("Recieved Initial Packet from %s\n", inet_ntoa(mediumAddr.sin_addr));
108             printf("Sequence Number #d - \n %s : \n", packet_recv.seqNum, packet_recv.data);
109             strcat(buffer, packet_recv.data);
110             base = packet_recv.id % 10;
111             short ack = packet_recv.acq;
112         }
```

Ensuite, on va envoyer les acquittements correspondants aux N segments de données envoyés. Nous allons vérifier aussi si tout les données ont été renvoyées de la part du client.



```
134
135 /* Renvoi de l'acquittement pour le paquet reçu */
136 if (packet_recv.type==16 && base >= 0 && base < packet_send.fenetre){
137     printf("-Sending ACKK #d\n", packet_send.id);
138     if (sendto(sockfd, &packet_send, sizeof(packet_send), 0,
139         (struct sockaddr *) &mediumAddr, sizeof(mediumAddr)) != sizeof(packet_send))
140         printf("Error\n");
141 } else if (base == -1 ) {
142     printf("Recieved Teardown Packet\n");
143     printf("Sending Terminal ACK\n");
144     if (sendto(sockfd, &packet_send, sizeof(Packet), 0,
145         (struct sockaddr *) &mediumAddr, sizeof(mediumAddr)) != sizeof(Packet))
146         printf("Error\n");
147 }
```

À la fin, le client est censé de recevoir l'acquittement pour chaque segment de données envoyé initialement par lui-même. Ensuite, nous allons fermer la connexion en utilisant le 3 way handshake pour le fermeture de la connexion entre le client et le server.



```
142
143 printf("Waiting for the ACK number\n");
144 originAddrLen = sizeof(originAddr);
145 while (respStringLen = recvfrom(sockfd, &packet_recv, sizeof(Packet), 0,
146     (struct sockaddr *) &originAddr, &originAddrLen) != -1)
147 {
148     printf("Received Packet : ACK No: %d\n", packet_recv.acq);
149     printf("Num Seq=%d\n", packet_recv.id);
150     if ((packet_recv.id%10)==9) break;
151 }
152
153 idValue--;
154 indexBuffer++;
155
```

Figure 1: Reception des acquittements par le client

```

160 Packet packet_fin;
161 packet_fin.type=2;
162 packet_fin.seqNum=92;
163 printf("FIN packet sent\n");
164
165 int closeConnection=0;
166 int closeFinACKLen;
167 while(closeConnection==0){
168     if (sendto(sockfd,&packet_fin,sizeof(Packet), 0, (struct sockaddr *)&destAddr,sizeof(destAddr)) != sizeof(packet_ser
169         printf("Erreur de l'envoi\n");
170     int packetFinSeq=packet_fin.seqNum;
171     printf("PacketFinSeqNum: %d\n",packetFinSeq);
172     while(1){
173         printf("Waiting for FIN-ACK\n");
174         if(closeFinACKLen = recvfrom(sockfd, &packet_fin, sizeof(Packet), 0,
175             (struct sockaddr *) &originAddr, &originAddrLen) == -1)
176             printf("[-] Error");
177         printf("Packet fin: acq=%d seqnum=%d\n",packet_fin.acq,packet_fin.seqNum);
178         if(packet_fin.acq==packetFinSeq+1 && packet_fin.type==18){
179             packet_fin.acq=packet_fin.seqNum+1;
180             if (sendto(sockfd,&packet_fin,sizeof(Packet), 0, (struct sockaddr *)&destAddr,sizeof(destAddr)) != sizeof(pa
181                 printf("Erreur de l'envoi\n");
182                 closeConnection=1;
183                 printf("Disconnecting...\n");
184                 sleep(2);
185                 break;
186     }
187 }

```

Figure 2: Fermeture de la connexion à l'aide de 3 way handshake

Données utilisées

Dans le struct.h, nous avons défini une structure qui définit le type de paquets que nous allons envoyer entre le client et le serveur. La définition a été faite ainsi:

```
typedef struct packet{
    char id;
    char type;
    short seqNum;
    short acq;
    char ecn;
    char fenetre;
    char data[44];
    int length;
} Packet;
```

Nous allons décrire ce que chaque champ représente:

- *id*: l'id d'un paquet;
- *type*: la valeur équivalent du type de paquet (par exemple 1 – SYN, 2 – FIN);
- *seqNum*: le nombre de séquence d'un paquet;
- *acq*: le nombre d'acquittement;
- *ecn*: la variable pour le contrôle de gestion;
- *fenetre*: le nombre de fenêtres pour chaque message que nous allons envoyer;
- *data[44]*: le contenu du message contenu dans les paquets envoyés;
- *length*: la longueur de data de chaque paquet.

Nous avons aussi utiliser la structure socket pour établir la connexion entre les éléments présents, c'est-à-dire le client, le medium et le serveur, ainsi que l'établissement des paramètres de connexion et la fonction *bind*:

```
struct sockaddr_in originAddr, destAddr;
socklen_t addr_size;

memset(&destAddr, 0, sizeof(destAddr));

originAddr.sin_family = AF_INET;
originAddr.sin_port=htons(port_local);
originAddr.sin_addr.s_addr= inet_addr("127.0.0.1");

destAddr.sin_family=AF_INET;
destAddr.sin_port=htons(port_dest);
destAddr.sin_addr.s_addr= inet_addr("127.0.0.1");

bind(sockfd, (const struct sockaddr*)&originAddr, sizeof(struct sockaddr_in));
```

, suivie de sa fermeture à l'aide de la fonction *close(sockfd)*.

Structure du projet

- *mainSender.c* qui contient le point principale d'exécution pour tous Sender, c'est-à-dire les fichiers qui font la simulation du client;
- *GoBackNSender* et *GoBackNReceiver* qui représentent le client et le serveur dans la simulation de la communication qui utilise le protocole *Go-Back-N*;
- *SWSender* et *SWReceiver* qui représentent le client et le serveur dans la simulation de la communication qui utilise le protocole *Stop and Wait*;
- *struct.h* qui contient la définition du type des paquets que nous allons envoyer;
- *Makefile*

Exécution du projet

À cause du manque de temps pour proprement préparer un makefile fonctionnel qui répond aux besoin exprimés dans l'énoncé, nous utilisons un makefile qui crée les objets exécutable associé à chaque fichier .c du projet et pour simuler la communication entre le client et le serveur géré par le perturbateur, nous ouvrons les 3 terminals suivants:

- un premier où nous allons exécuter l'objet exécutable du client;

Exemple: `gcc mainSender.c -o main`

`./main 1 127.0.0.1 3333 4444`

(où 1 est le nombre qui correspond au protocole Go-Back-N, 127.0.0.1 est l'adresse IP distant, 3333 est le port du client et 4444 le port du perturbateur)

Si le deuxième paramètre est égal à 2, cela veut dire que nous voulons simuler le protocole Stop And Wait);

- un deuxième où nous allons exécuter le fichier `medium.py` avec les flags `-s -v` pour suivre l'envoi des paquets;

- un troisième où nous allons exécuter l'objet exécutable du serveur;

Exemple: `gcc GoBackNReceiver -o GBNServer`

`./GBNServer 127.0.0.1 6666 5555`

(où 127.0.0.1 est l'adresse IP distant, 6666 est le port du serveur et 5555 le port du perturbateur).

Conclusion

En tenant compte des résultats obtenus pendant les tests, nous pouvons conclure que le protocole **Go-Back-N** est plus efficace que **Stop and Wait**.