Projet C
—
Commutateur niveau 3

Philippe TRAN BA - Élie BOUTTIER - Jiajun SHI - Émilie ABIA

ENSEEIHT, département TR

13 juin 2012

1. Introduction
2. Architecture du programme
3. Implémentation
   - Trame
   - PacketFrag
   - ListPacketFrag
   - Packet
   - Commut
   - Queue
   - Simulator
4. Difficultées et améliorations
   - Difficultées
   - Améliorations
5. Conclusion

Plan

1 Introduction

2 Architecture du programme

3 Implémentation

4 Difficultées et améliorations

5 Conclusion

Plan

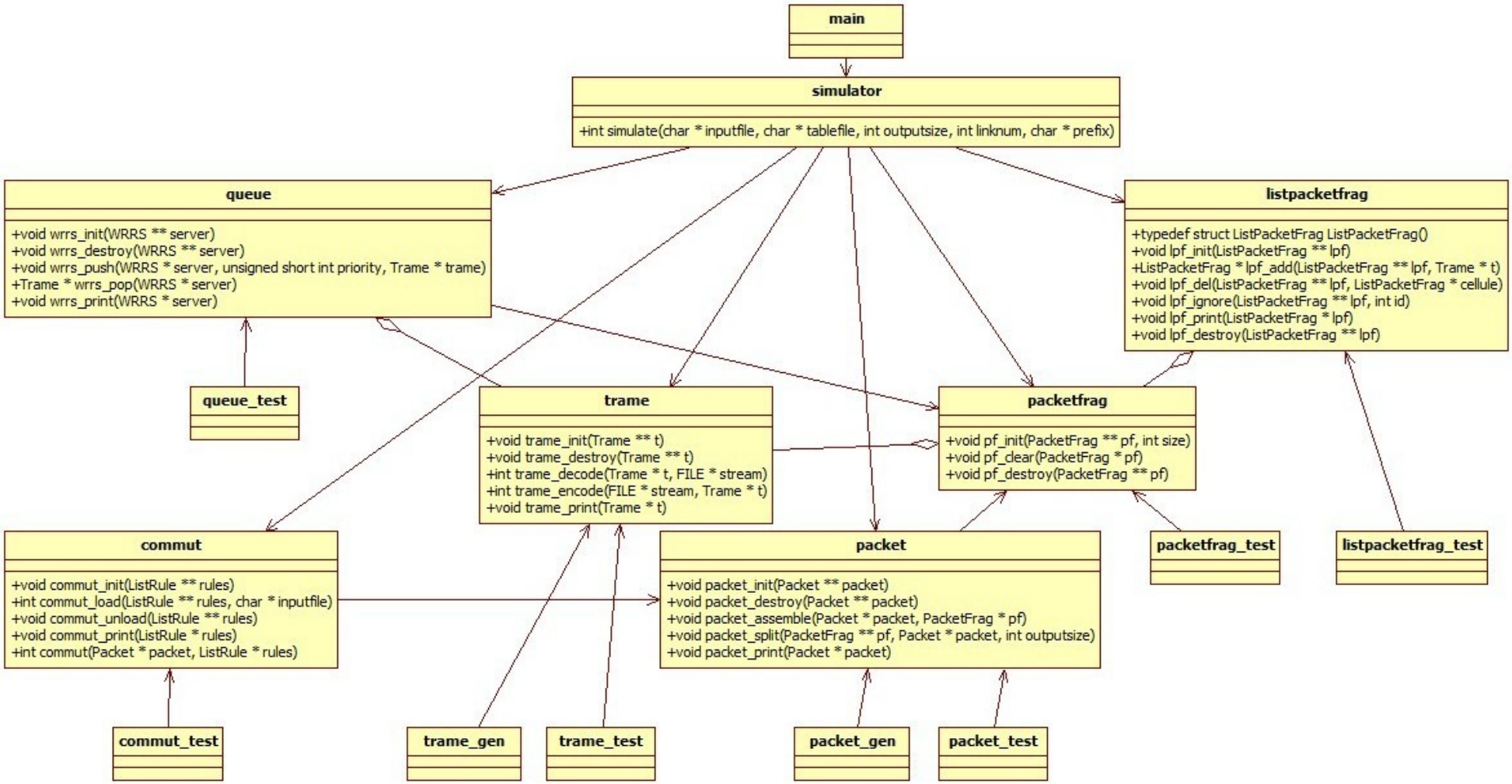
1 Introduction

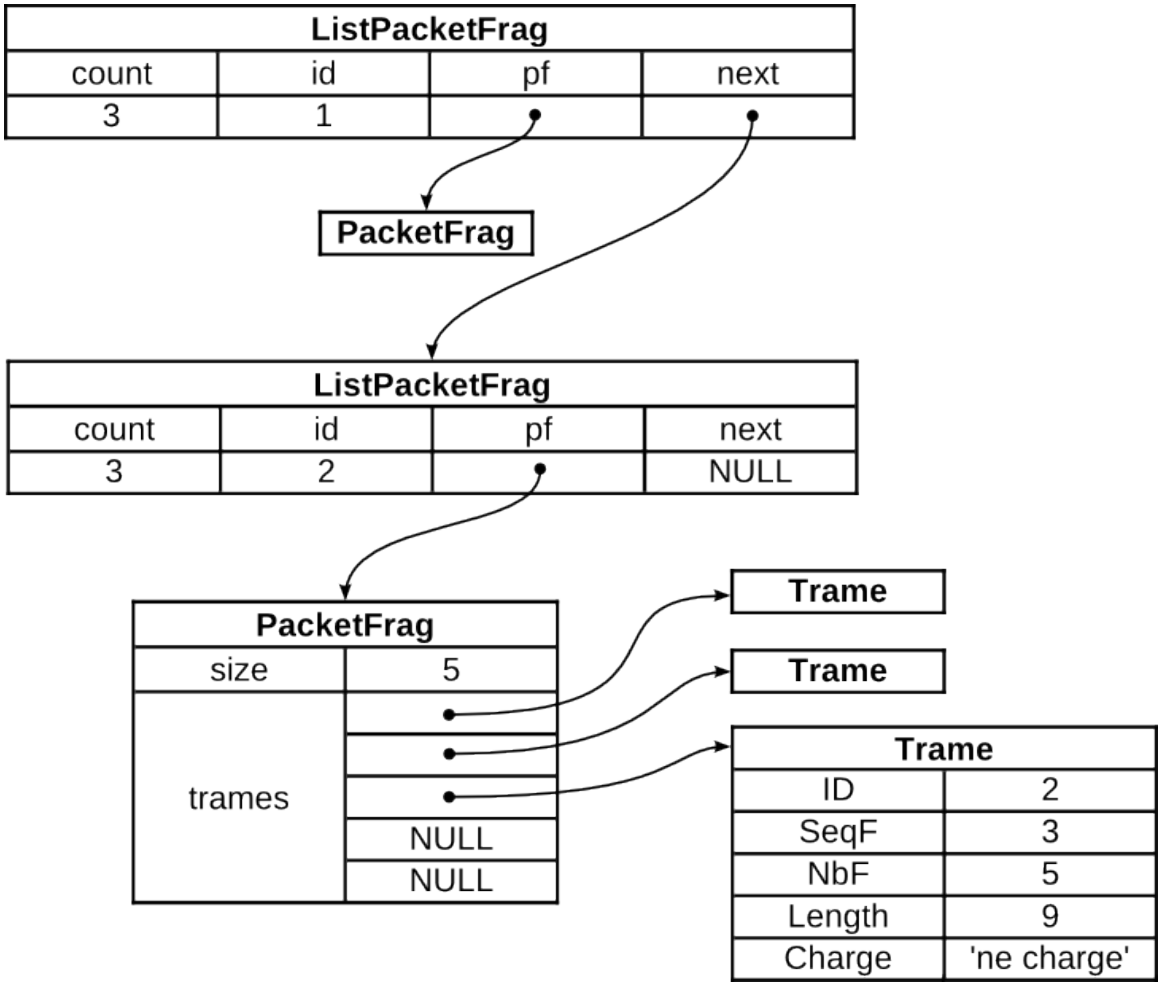
2 Architecture du programme

3 Implémentation

4 Difficultées et améliorations

5 Conclusion


main
simulator
+int simulate(char * inputfile, char * tablefile, int outputsize, int linknum, char * prefix)
queue
+void wrrs_init(WRRS ** server)
+void wrrs_destroy(WRRS ** server)
+void wrrs_push(WRRS * server, unsigned short int priority, Trame * trame)
+Trame * wrrs_pop(WRRS * server)
+void wrrs_print(WRRS * server)
listpacketfrag
+typedef struct ListPacketFrag ListPacketFrag()
+void lpf_init(ListPacketFrag ** lpf)
+ListPacketFrag * lpf_add(ListPacketFrag ** lpf, Trame * t)
+void lpf_del(ListPacketFrag ** lpf, ListPacketFrag * cellule)
+void lpf_ignore(ListPacketFrag ** lpf, int id)
+void lpf_print(ListPacketFrag * lpf)
+void lpf_destroy(ListPacketFrag ** lpf)
queue_test
trame
+void trame_init(Trame ** t)
+void trame_destroy(Trame ** t)
+int trame_decode(Trame * t, FILE * stream)
+int trame_encode(FILE * stream, Trame * t)
+void trame_print(Trame * t)
packetfrag
+void pf_init(PacketFrag ** pf, int size)
+void pf_clear(PacketFrag * pf)
+void pf_destroy(PacketFrag ** pf)
commut
+void commut_init(ListRule ** rules)
+int commut_load(ListRule ** rules, char * inputfile)
+void commut_unload(ListRule ** rules)
+void commut_print(ListRule * rules)
+int commut(Packet * packet, ListRule * rules)
packet
+void packet_init(Packet ** packet)
+void packet_destroy(Packet ** packet)
+void packet_assemble(Packet * packet, PacketFrag * pf)
+void packet_split(PacketFrag ** pf, Packet * packet, int outputsize)
+void packet_print(Packet * packet)
packetfrag_test
listpacketfrag_test
commut_test
trame_gen
trame_test
packet_gen
packet_test


ListPacketFrag
count
id
pf
next
3
1
PacketFrag
ListPacketFrag
count
id
pf
next
3
2
NULL
PacketFrag
size
5
trames
NULL
NULL
Trame
Trame
Trame
ID
2
SeqF
3
NbF
5
Length
9
Charge
'ne charge'

Plan

1 Introduction

2 Architecture du programme

3 Implémentation

4 Difficultées et améliorations

5 Conclusion

```c
/* Structure d'une trame */
typedef struct {
  unsigned short int id; /* numero du paquet */
  unsigned short int seqf; /* numero de sequence */
  unsigned short int nbf; /* nombre de sequence */
  unsigned int length; /* longeur de charge */
  unsigned char * charge; /* contenue */
} Trame;

/* Initialiser et detruire une trame */
void trame_init(Trame ** t);
void trame_destroy(Trame ** t);

/* Decoder et encoder une trame */
int trame_decode(Trame * t, FILE * stream);
int trame_encode(FILE * stream, Trame * t);
```

```
1  while (state < 9 && state > -1) {
2      pending = fgetc(stream); /* Lecture d'un caractere */
3      if (pending == EOF) {
4          fprintf(stderr, "trame_decode: error in fgetc\n");
5          state = -1;
6      } else {
7          switch (state) {
8              case 0: // Attente du premier fanion
9                  if (pending == FANION) {
10                     state++; /* On passe a l'etat suivant */
11                 } else {
12                     state = -1;
13                 }
14                 break;
15         }
16     }
17 }
```

```c
/* PacketFrag, utilise pour stocker les trame d'un meme packet
struct PacketFrag {
    // Nombre de fragment (taille du tableau)
    int size;
    // Tableau contenant les trames
    Trame ** trames;
};
typedef struct PacketFrag PacketFrag;

void pf_init(PacketFrag ** pf, int size);

void pf_clear(PacketFrag * pf);
void pf_destroy(PacketFrag ** pf);
```

```c
typedef struct {
    // Nombre de trame stocke dans le packet fragmente
    int count;
    // ID du packet associe
    int id;
    // Packet fragmente de la cellule
    PacketFrag * pf;
    // Cellule suivante
    struct ListPacketFrag * next;
} ListPacketFrag;

/* Ajouter une trame dans la structure */
ListPacketFrag * lpf_add(ListPacketFrag ** lpf, Trame * t);
/* Supprimer un packet fragmente de la liste. */
void lpf_del(ListPacketFrag ** lpf, ListPacketFrag * cellule);
/* Ignorer l'ajout de toutes les trames de l'ID specifie */
void lpf_ignore(ListPacketFrag ** lpf, int id);
```

```c
typedef struct {
  unsigned int label;
  unsigned char priority;
  unsigned int size;
  unsigned char * charge;
} Packet;

/* Assembler un packet */
void packet_assemble(Packet * packet, PacketFrag * pf);

/* Fragmenter un packet */
void packet_split(PacketFrag ** pf, Packet * packet,
        int outputsize);
```

```c
/* Liste chaine des regles de commutation */
struct ListRule {
    unsigned int inlabel;
    unsigned int outlabel;
    int outlink;
    struct ListRule * next;
};

typedef struct ListRule ListRule;

/* Charger les regles de routage depuis un fichier */
int commut_load(ListRule ** rules, char * inputfile);

/* Commuter un packet.
 * Change le label et renvoit le lien de sortie.
 * Renvoit -1 si le packet n'est concerne par aucune regle. */
int commut(Packet * packet, ListRule * rules);
```

```c
/* Liste chaine circulaire */
typedef struct {
    struct Queue * previous;
    Trame * element;
} Queue;

typedef struct { /* Weighted Rount Robin Server */
    Queue * queue[NBPRIORITY]; // Tableau des queues
    int priority; // Priorite en cours
    int credit; // Nombre de credits restants
} WRRS;

/* Ajouter une trame dans le serveur */
void wrrs_push(WRRS * server, unsigned short int priority,
        Trame * trame);
/* Obtenir la prochaine trame devant sortir du serveur. */
Trame * wrrs_pop(WRRS * server);
```

```
1  int simulate(char * inputfile, char * tablefile, int outputsize
2                  int linknum, char * prefix)
3  {
4      commut_load(&rules, tablefile);
5      servers = malloc(linknum * sizeof(*servers));
6      for (i = 0; i < linknum; i++) {
7          wrrs_init(servers+i); }
8          stream = fopen(inputfile, "r");
9      while ((pending = fgetc(stream)) != EOF && state != -1) {
10         switch (state) {
11             // (...)
12         }
13     }
14     commut_unload(&rules);
15         for (i = 0; i < linknum; i++) {
16         wrrs_destroy(servers+i); }
17 }
```

```
1  // Fonction de traitement sur un IN
2  int in(FILE * stream, ListPacketFrag ** list, ListRule * rules
3          int outputsize, WRRS ** servers) {
4      trame_decode(t, stream); // Decodage de la trame
5          // Ajout de la trame a la structure
6          lpfdone = lpf_add(list, t);
7      if (lpfdone == NULL) { return 0; }// Arret si paquet incom-
   plet
8          packet_assemble(packet, lpfdone->pf); // Assemblage
9      // Routage
10     if ((outlink = commut(packet, rules)) < 0) { return 0; }
11         // Fragmentation du packet
12         packet_split(&packetfrag, packet, outputsize);
13         // Ajout des trames dans la queue approprié
14         for (i = 0; i < packetfrag->size; i++) {
15                 wrrs_push(servers[outlink], packet->priority,
16         }
17 }
```

```c
int out(WRRS ** servers, int link, int linknum, char * prefix)
{
    static FILE ** streams = NULL; // Fichier de sortie
    // Allocation du tableau de flux, initialise a zero
    if (streams == NULL) {
        streams = calloc(linknum, sizeof(*streams));
    }
    Trame * t = wrrs_pop(servers[link]);
    if (streams[link] == NULL) { // Le fichier n'a pas encore
ete ouvert
        sprintf(filename, "%s-%d", prefix, link);
        if ((streams[link] = fopen(filename, "a")) == NULL) {
            return -1;
        }
    }
    trame_encode(streams[link], t);
    return 0;
}
```

Plan

1 Introduction

2 Architecture du programme

3 Implémentation

4 Difficultées et améliorations

5 Conclusion

- Lecture et ≪ décodage à la volé ≫ des trames
- Liste de liste de trame
- Ignorer les paquets dont la première trame reçu est erronée
- Calcul de la CRC

- Gestion des erreurs
- Asynchronisation des entrées/sorties

Plan

1 Introduction

2 Architecture du programme

3 Implémentation

4 Difficultées et améliorations

5 Conclusion