Projet C
—
Commutateur niveau 3

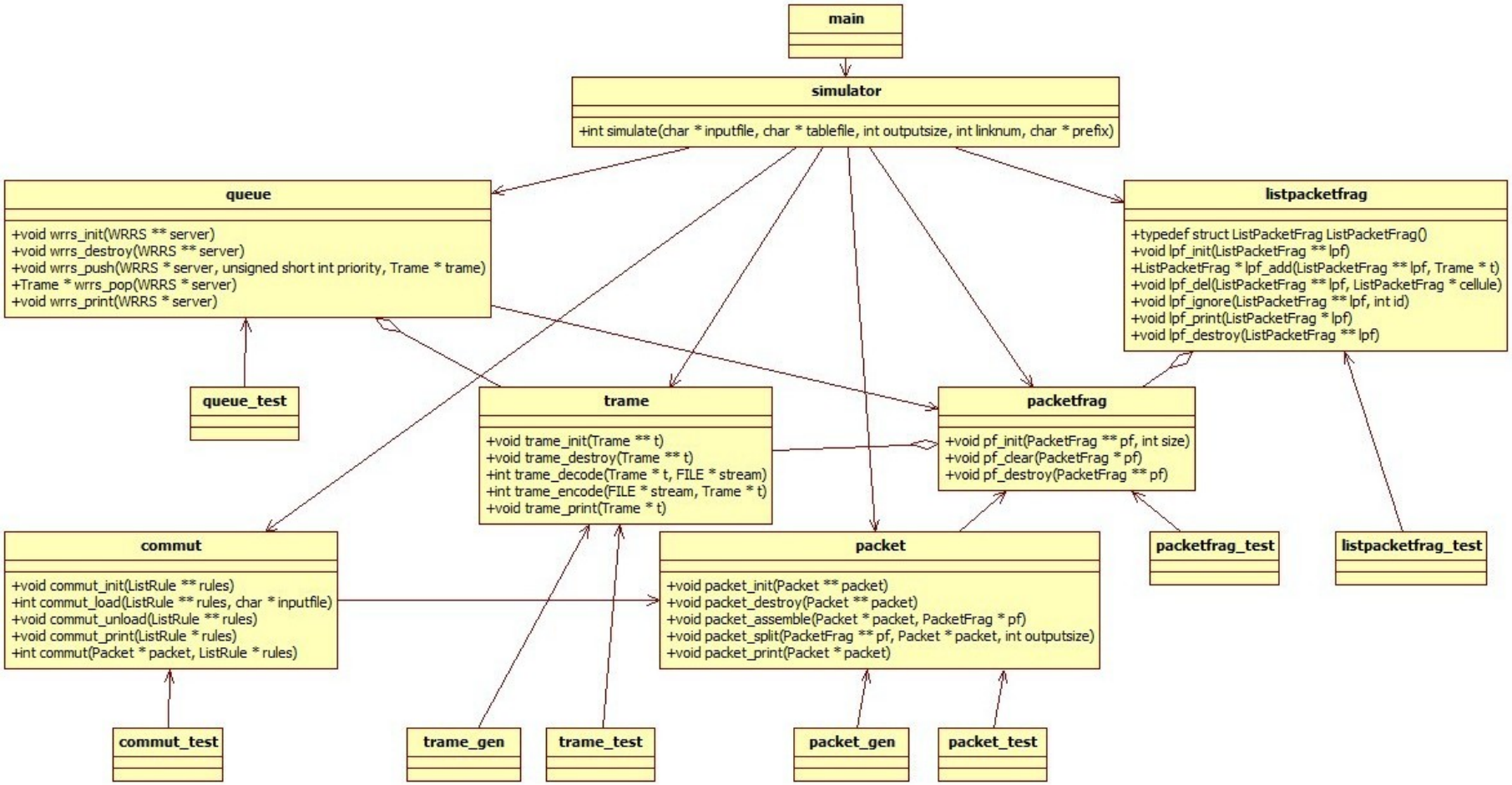
Philippe TRAN BA - Élie BOUTTIER - Jiajun SHI - Émilie ABIA

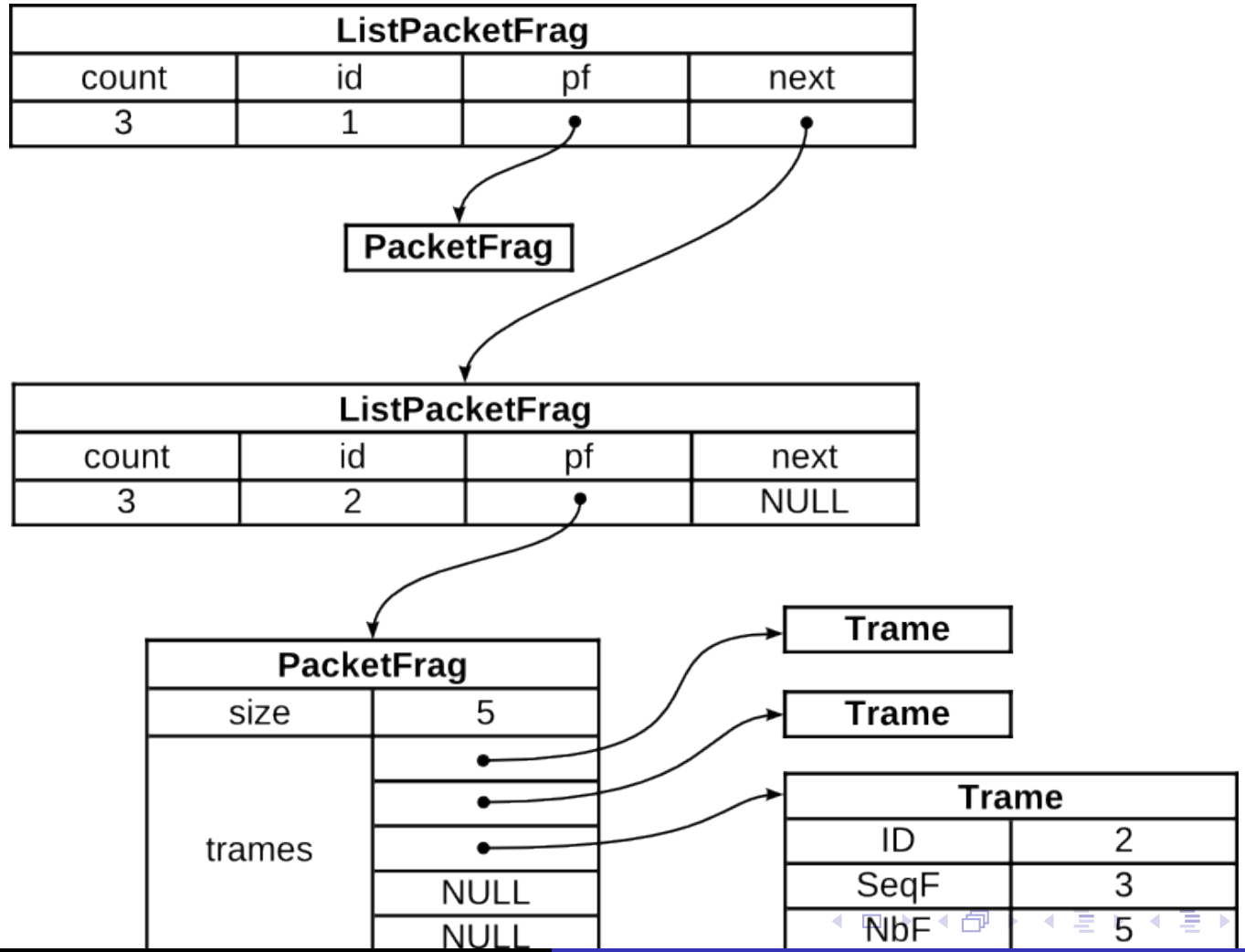
ENSEEIHT, département TR

13 juin 2012

Plan

Plan


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

Plan

```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              // Attente du premier fanion
9              case 0:
10                 if (pending == FANION) {
11                     state++; /* On passe a l'etat suivant */
12                 } else {
13                     fprintf(stderr, "trame_decode: [...]\n");
14                     state = -1;
15                 }
16                 break;
17         }
18     }
```

```
1  /* PacketFrag, utilise pour stocker les trame d'un meme packet
2  struct PacketFrag {
3      // Nombre de fragment (taille du tableau)
4      int size;
5      // Tableau contenant les trames
6      Trame ** trames;
7  };
8  typedef struct PacketFrag PacketFrag;
9
10 void pf_init(PacketFrag ** pf, int size);
11
12 void pf_clear(PacketFrag * pf);
13 void pf_destroy(PacketFrag ** pf);
```

```c
struct ListPacketFrag {
    // 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;
};
typedef struct ListPacketFrag 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);
```

```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);
```

```
1  /* Liste chaine circulaire */
2  typedef struct {
3      struct Queue * previous;
4      Trame * element;
5  } Queue;
6
7  typedef struct { /* Weighted Rount Robin Server */
8      Queue * queue[NBPRIORITY]; // Tableau des queues
9      int priority; // Priorite en cours
10     int credit; // Nombre de credits restants
11 } WRRS;
12
13 /* Ajouter une trame dans le serveur */
14 void wrrs_push(WRRS * server, unsigned short int priority,
15         Trame * trame);
16
17 /* Obtenir la prochaine trame devant sortir du serveur.
18  * Retourne NULL si le server est vide. */
```

```
1  int main(int argc, char ** argv)
2  {
3          char * inputfile = DEFAULT_INPUT_FILE;
4      // (...)
5
6          for (opt = 1 ; opt < argc && err == 0 ; opt++) {
7                  cmd = argv[opt];
8                  if (!strcmp(cmd, "-i")) {
9              // (...)
10                 } else {
11                         err = 2; /* unknow argument */
12                 }
13         }
14
15         if (err != 0) { exit(EXIT_FAILURE); }
16
17         if (simulate(inputfile, tablefile, outputsize,
18                 linknum, prefix) < 0) { exit(EXIT_FAILURE);
```

```
1  // Lancer la simulation
2  int simulate(char * inputfile, char * tablefile, int outputsize
3                  int linknum, char * prefix) {
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
10     while ((pending = fgetc(stream)) != EOF && state != -1) {
11         switch (state) {
12             // (...)
13         }
14     }
15
16     commut_unload(&rules);
17         for (i = 0; i < linknum; i++) {
18         wrrs_destroy(servers+i); }
```

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

```
1  // Fonction de traitement sur un OUT
2  int out(WRRS ** servers, int link, int linknum, char * prefix)
3  {
4      static FILE ** streams = NULL; // Fichier de sortie
5      // Allocation du tableau de flux, initialise a zero
6      if (streams == NULL) {
7          streams = calloc(linknum, sizeof(*streams));
8      }
9      Trame * t = wrrs_pop(servers[link]);
10     if (streams[link] == NULL) { // Le fichier n'a pas encore
   ete ouvert
11         sprintf(filename, "%s-%d", prefix, link);
12         if ((streams[link] = fopen(filename, "a")) == NULL) {
13             return -1;
14         }
15     }
16     trame_encode(streams[link], t);
17     return 0;
```

Plan