INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES RENNES

Domitille BUTTIN

Valentin CAILLARD

Antoine LE CALVEZ

Thibault PEREL

Projet Langage C

Simulateur de réseaux

3 SRC 2014-2015

Table des matières

Introduction

Dans le cadre du module Informatique de troisième année du département Systèmes et Réseaux de Communication (SRC), nous avons réalisé par groupe de quatre personnes un projet informatique codé en Langage C. Nous avons choisi le sujet proposé par Mme F. NOUVEL :

« **Réaliser un simulateur de réseaux** »

Ce sujet nous a paru intéressant car il s'agissait de créer un logiciel destiné à expliquer le fonctionnement des réseaux. Ce simulateur a pour but de modéliser le comportement des éléments de la structure (nœuds, PC) en fonction des paramètres configurés par l'utilisateur. L'enjeu était donc de créer une interface graphique la plus intuitive possible afin d'expliquer aux non-initiés le fonctionnement de ce type de réseau.

Ce projet avait pour intérêt de nous familiariser avec l'environnement graphique codé en Langage C tout en s'appropriant des notions de base sur les communications en réseaux, thème que nous n'abordons qu'en quatrième année du département SRC.

Dans un premier temps, nous présenterons plus en détail le cahier des charges ainsi que les différents objectifs du projet. Puis, nous analyserons le programme avec ses étapes de conception et de programmation. Nous expliquerons également le fonctionnement du logiciel. Enfin, nous conclurons sur les différents problèmes rencontrés ainsi que sur les améliorations possibles qui peuvent être apportées au logiciel.

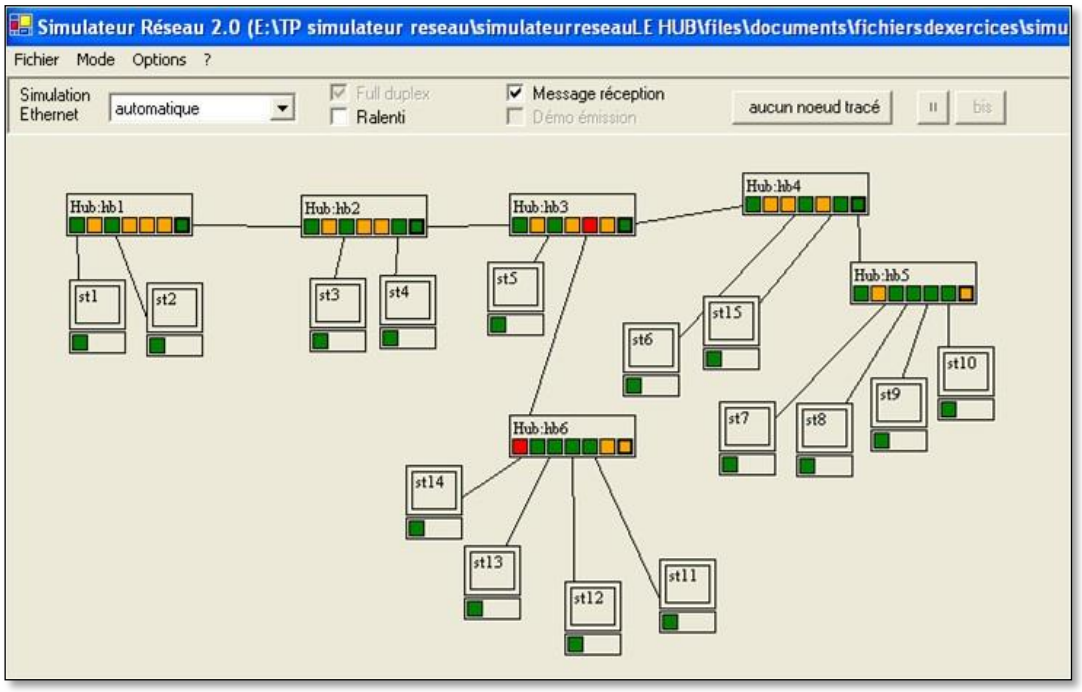
I. Présentation du sujet et cahier des charges

Comme nous l'avons expliqué en introduction, le but de ce projet était de réaliser un simulateur de réseaux. Nous allons maintenant établir les critères que notre logiciel devait respecter pour modéliser le plus fidèlement possible le comportement de ces réseaux.

Pour communiquer des données entre deux postes, il faut nécessairement un poste émetteur, un poste récepteur, des données à envoyer et un canal de transmission. Les informations sont encapsulées dans des trames. Pour transmettre ensuite ces trames, il n'y a pas de chemin établi au préalable. Le protocole est « non orienté connexion ». Il y a donc un caractère aléatoire dans la transmission de données. La plupart du temps, le message doit être alors remis en ordre.

De plus, il se peut que des trames se perdent durant leur transmission. Il arrive qu'il y ait collision avec d'autres trames envoyées en simultané. Le but d'un simulateur de réseau est de reproduire ces chemins aléatoires et de gérer les collisions et conflits que l'on peut trouver au sein d'un réseau.

Le cahier des charges a été établi en accord avec Mme F. NOUVEL et nous avions comme modèle le logiciel de Mr P. LOISEL.



Un des objectifs était de simuler l'envoi de plusieurs paquets depuis divers ordinateurs sources jusqu'aux récepteurs. Cette condition devait nous amener à gérer la collision entre les paquets.

Un autre objectif était de rendre compte de la réception de plusieurs trames. En effet, si les trames étaient reçues dans le désordre, il faudrait alors procéder à une remise en ordre du message. Nous devions être également capables d'ajuster le débit, la longueur des fils, le nombre de trames émises et de calculer le temps de transmission total du message.

De plus, le simulateur présenterait une architecture de base, composée de nœuds et de PC. Un des objectifs était de faire évoluer cette architecture, en ajoutant ou supprimant des éléments du réseau.

Enfin, nous avions la contrainte de gérer aléatoirement le déplacement des paquets, tout en les faisant arriver à la bonne destination par la communication et la gestion des adresses de chaque PC.

II. Conception du logiciel

Dans cette partie, nous allons d'abord expliquer notre choix concernant l'environnement de travail, puis nous détaillerons les différentes étapes de programmation qui nous ont amenés à réaliser le simulateur.

1. Environnement de travail

a) IDE : CodeBlocks et simulateur sur console

La première étape dans le processus de programmation fut d'établir un simulateur sur console grâce au logiciel CodeBlocks, dans l'idée future de faire le lien avec l'interface graphique.

Nous avons alors construit 6 structures :

- *PC* servant d'émetteur/récepteur du message ;
- *Switch* constituant les différents nœuds ;
- *Fil* pour les caractéristiques des fils (débit, longueur) ;
- *Message* comportant les caractéristiques du message à transmettre ;
- *Trame* caractérisant chaque élément du message à transmettre ;
- *Réseau* comprenant la description globale du réseau courant.

Une fois ces structures définies, nous avons réalisé des fonctions permettant de transmettre des messages d'un émetteur vers un destinataire. Après réception, d'autres fonctions étaient destinées à remettre en ordre les données reçues et de les afficher.

De plus, des fonctions d'édition du réseau étaient alors disponibles comme : modifier la longueur des fils, ajuster leur débit, ajouter ou supprimer des PC et des nœuds. Enfin, notre « main » contenait essentiellement une fonction de type « switch-case » qui répertoriait toutes les actions que l'utilisateur pouvait effectuer. Une fois ces fonctions mises au point, nous nous sommes alors intéressés à la partie graphique du programme.

b) Bibliothèques GTK/ SDL

La première étape de la programmation graphique fut de choisir la bibliothèque. Dans un premier temps, nous avons choisi GTK. Nous nous sommes rendu compte que cette bibliothèque graphique était peu adaptée à nos besoins. En effet, nous avions besoin d'affichage dynamique et de gérer des images, choses qui sont gérées plus efficacement avec la bibliothèque SDL. En plus de cette dernière, nous avons utilisé les bibliothèques SDL_image pour la gestion des formats d'image et SDL_ttf pour l'affichage de texte à l'écran.

2. Etapes de programmation de l'interface graphique

Une fois la bibliothèque choisie, nous avons travaillé sur l'interface graphique du logiciel. Ayant comme base un simulateur fonctionnant sous console, nous avons d'abord cherché à joindre nos fonctions avec l'interface. Cependant, nous nous sommes rendu compte qu'il n'était pas possible de fusionner les deux. Nous avons dû alors programmer à nouveau notre simulateur de manière graphique cette fois. Cette partie s'est déroulée en trois grandes étapes :
- l'initialisation avec le positionnement des éléments fixes ;
- le choix des paramètres et la gestion des évènements ;
- l'envoi des paquets d'un PC source à un PC destinataire.

a) Initialisation des éléments fixes

Cette première étape a consisté à charger la bibliothèque SDL, définir la fenêtre du logiciel puis à créer les différentes surfaces de travail.

En premier lieu, il s'agissait de charger le système d'affichage de la bibliothèque SDL et d'effectuer un test afin de savoir si la charge s'était bien déroulée.

```
//Initialisation de SDL
if(SDL_Init(SDL_INIT_VIDEO)==-1)
{
    printf("Erreur d'initialisation de SDL: %s\n", SDL_GetError());
    exit(EXIT_FAILURE);
}
```

Ensuite, il a fallu créer la fenêtre de travail en définissant : le fond, le nom, les dimensions et éventuellement quelques options comme le réajustement de la fenêtre.

```
//Icone de la fenetre
  icone = IMG_Load("icone.png");
  SDL_WM_SetIcon(icone, NULL);
```

```
//Paramètres de la fenêtre
  ecran = SDL_SetVideoMode(SCREEN_WIDTH,
                           SCREEN_HEIGHT,
                           SCREEN_BPP,
                           SDL_HWSURFACE | SDL_RESIZABLE);
//Nom de la fenêtre
  SDL_WM_SetCaption("Network simulator", NULL);
```

Vient finalement la création de « surfaces ». Ce sont des zones de l'interface que l'on va pouvoir gérer individuellement. Voici quelques exemples d'utilisation des surfaces :
- les boutons : run, cancel, reboot, plus, moins, sauvegarde, etc ;
- les icônes : noeuds et PC ;
- les paramètres : vitesse de transmission et nombre de trames à envoyer.

Mais comment est créée la surface ? Nous allons uniquement prendre l'exemple de la surface destinée au bouton « run/cancel ». Toutes les surfaces citées précédemment ont été programmées de manière analogue.

Il a fallu d'abord initialiser les paramètres de cette surface définie par ses coordonnées x et y, sa largeur et enfin sa hauteur.

```
   // Pointeur de la surface
SDL_Surface *run = NULL;

//Les paramètres du bouton run/cancel
#define RUN_CANCEL_X 15
#define RUN_CANCEL_Y 700
#define RUN_CANCEL_WIDTH 110
#define RUN_CANCEL_HEIGHT 110
```

Nous avons procédé à **l'initialisation de la position du bouton** avec la fonction « init_position » :

```
//Position de l'image du bouton run/cancel
   SDL_Rect positionRunCancel;
   init_position(&positionRunCancel){
    (*positionRunCancel).x = RUN_CANCEL_X;
    (*positionRunCancel).y = RUN_CANCEL_Y;
    (*positionRunCancel).h = RUN_CANCEL_HEIGHT;
    (*positionRunCancel).w = RUN_CANCEL_WIDTH;
}
```

Ensuite vient le **découpage des sprites**. L'image au format « .png » que nous avons chargée dans le programme est la suivante :

Nous avons pu passer du bouton vert au bouton rouge grâce à l'utilisation du « sprite » de la bibliothèque SDL. Il s'agit tout simplement de définir quelle partie de l'image nous souhaitons afficher. Cela s'apparente à rogner une image. Il a donc fallu jouer sur les coordonnées de la feuille de sprite de cette image pour ensuite soit afficher le bouton rouge pendant la simulation soit le bouton vert le reste du temps.

```
//Les paramètres de la feuille de sprite Run/Cancel
#define RUN_CANCEL_WIDTH_SPRITE 220
#define RUN_CANCEL_HEIGHT_SPRITE 110

//Tableau de positions de la feuille de sprite pour le run/cancel
   SDL_Rect run_cancel_sprite[ 2 ];

// Découpage de la feuille de sprite :
// run_cancel_sprite[0] = bouton vert
// run_cancel_sprite[1] = bouton rouge

init_sprite(run_cancel_sprite){
   // run_cancel_sprite : On coupe la partie en haut à droite (bouton run)
   run_cancel_sprite[ 0 ].x = 0;
   run_cancel_sprite[ 0 ].y = 0;
   run_cancel_sprite[ 0 ].w = RUN_CANCEL_WIDTH_SPRITE/2;
   run_cancel_sprite[ 0 ].h = RUN_CANCEL_HEIGHT_SPRITE;
   //run_cancel_sprite : On coupe la partie en haut à gauche (bouton cancel)
   run_cancel_sprite[ 1 ].x = RUN_CANCEL_WIDTH_SPRITE/2;
   run_cancel_sprite[ 1 ].y = 0;
   run_cancel_sprite[ 1 ].w = RUN_CANCEL_WIDTH_SPRITE/2;
   run_cancel_sprite[ 1 ].h = RUN_CANCEL_HEIGHT_SPRITE;}
```

Finalement, nous n'avions plus qu'à **charger l'image** du dossier courant dans le programme à l'aide de la fonction « init_charger_images ».

```
//Chargement des images
   init_charger_images(&run){
*run = IMG_Load("run.png");
}
```

Puis à « blitter », c'est-à-dire **afficher le bouton** :

```
//Affichage des boutons
   init_blitter_boutons(&run,run_cancel_sprite){
//par défaut le bouton est vert
   apply_surface(    RUN_CANCEL_X,
                     RUN_CANCEL_Y,
                     *run,
                     *ecran,
                     &run_cancel_sprite[ 0 ] );

}
```

b) Paramétrage et gestion des événements

Une fois l'initialisation terminée, nous avons alors programmé les paramètres du simulateur :

- le nombre de paquets à envoyer ;
- le débit ;
- le nombre de PC sur le réseau ;
- les PC source et destinataire.

La première partie a consisté à ajouter et supprimer des ordinateurs. Les coordonnées du clic de l'utilisateur définissent la position du nouveau PC dans le réseau.

Vient ensuite le choix du débit et du nombre de paquets envoyés par l'émetteur.

Enfin, nous avons programmé les fonctions liées à la simulation comme start, reboot et également les fonctions liées à la sauvegarde du réseau et à l'ouverture du réseau enregistré.

La gestion des évènements revient à anticiper tous les choix de l'utilisateur. Ainsi, nous avons répertorié tous les évènements que nous pensions possibles dans un « switch-case ». Une boucle « while » nous permet de sortir de la boucle uniquement si on clique sur la croix rouge.

```
//Fonction principale de gestion des évènements
    while (continuer)
    {//Tant que l'utilisateur veut continuer (pas de clic sur la croix rouge)
        //On attend un évènement
        SDL_WaitEvent(&event);
        switch(event.type)
        {
            //Si l'utilisateur clique sur la croix rouge
            case SDL_QUIT:
                //Mise à 0 de la variable de la boucle
                continuer = 0;
                break;

            //Si l'utilisateur clique sur un bouton de la souris
            case SDL_MOUSEBUTTONDOWN:
                if (event.button.button == SDL_BUTTON_LEFT)
                {//Si l'évènement est un clic gauche
                    //Récupération des coordonnées du clic
                    position_clic.x = event.motion.x;
                    position_clic.y = event.motion.y;
                        [...]
```

Si cette action n'est pas effectuée, le programme boucle à l'infini. L'avantage de cette technique est que l'ordinateur procède à une réactualisation du programme en continu.

Une grande partie du « main » est donc constitué de conditions, qui s'enclenchent en fonction des choix de l'utilisateur et qui entraînent alors une action du programme.

De plus, nous avons dissocié les actions "appuyer sur le bouton de la souris" et "relâcher le bouton de la souris". La gestion des évènements a donc été séparée en deux parties. Cette décomposition accorde au programme une certaine interactivité avec l'utilisateur. Nous précisons que tous les évènements concernent le clic gauche de la souris uniquement.

```
{//Si on clique sur le bouton ouvrir
        //On met en surbrilliance le bouton ouvrir
        apply_surface(  OUVRIR_X,
                        OUVRIR_Y,
                        ouvrir_sauvegarder,
                        ecran,
                        &ouvrir_sauvegarder_sprite[ 2 ] );
        //On indique que le bouton ouvrir a été cliqué
        bouton_ouvrir = 1;
}
```

c) Simulation de l'envoi des paquets

Une fois le réseau paramétré, on choisit un PC émetteur et un PC récepteur. Il est alors possible de lancer une simulation en cliquant sur le bouton « run ». Une enveloppe se déplace entre ces deux points. Elle symbolise la transmission des paquets.

En ce qui concerne le déplacement de cette enveloppe, le programme entre dans une boucle « while » qui est conditionné par le nombre de paquets transmis.

Au départ, l'enveloppe se place au niveau du PC émetteur. Tant que l'enveloppe n'a pas atteint le nœud le plus proche, une boucle décale petit à petit l'image, ceci en fonction de la vitesse de transmission choisie. Cela donne à l'utilisateur l'impression que l'enveloppe se déplace.

Lorsque l'enveloppe arrive à un nœud, nous avons programmé le test suivant :
- si l'enveloppe est arrivée sur le nœud où est relié le PC destinataire, alors elle se dirige vers celui-ci ;
- si le PC destinataire n'est pas connecté à ce nœud, alors l'enveloppe se déplace aléatoirement vers les autres nœuds.

Après avoir testé les différents nœuds sur lesquels l'enveloppe passe, celle-ci arrive obligatoirement à l'ordinateur destinataire. Seul le temps de transmission va varier suivant le chemin qu'elle va prendre.

Le calcul du temps de transmission est effectué par l'intermédiaire du fichier « time.h ». Lorsque l'envoi est terminé, la condition « envoi_termine » est vraie et le logiciel calcule alors le temps total de l'envoi et l'affiche à l'écran.

```
//Variables pour le calcul du temps d'envoi
   clock_t temps;
   double temps_envoi;
   char str[20];
//Variable pour détecter quand l'envoi est terminé
//et qu'il faut calculer le temps d'envoi
   int envoi_termine = 0;

if(envoi_termine)
    {//Si l'envoi est terminé, on calcule le temps d'envoi
        temps_envoi = (double) (clock()-temps)/CLOCKS_PER_SEC;
     //On convertit le nombre en chaine de caractère
        sprintf(str, "Time : %3.3fs", temps_envoi);
        texteTemps = TTF_RenderText_Blended(police, str, couleurNoire);
        SDL_BlitSurface(texteTemps,NULL,ecran,&positionTemps);
    }
```

Pendant la simulation, le logiciel affiche le nombre de paquets qui ont été réceptionnés. Pour cela, nous avons utilisé un « switch-case » qui est conditionné par le nombre de paquets sélectionnés par l'utilisateur. Pour chaque cas, le programme vérifie combien de paquets doivent être envoyés, il modifie alors le message « Packets reception : 1 » et affiche le texte à l'écran.

Voici le cas où il y a un seul paquet à envoyer. Dans le « else », nous avons pris en compte le fait qu'il y a peut-être d'autres paquets à envoyer, dans ce cas, le logiciel affiche « Packets reception : 1**…** ».

```
switch(taille_choisi-copie_taille){
    case 1: //Si un seul message a été envoyé
        if(taille_choisi==1)
            {//Si le nombre de message à envoyer est 1
                texte = TTF_RenderText_Blended(police, "Packets reception : 1", couleurNoire);
             //L'envoi est terminé
                envoi_termine = 1;
                texteOk = TTF_RenderText_Blended(police, "OK", couleurOk);
                SDL_BlitSurface(texteOk,NULL,ecran,&positionInfoOk);
            }
            //Si il faut envoyer plus d'un message
        else texte = TTF_RenderText_Blended(police, "Packets reception : 1..", couleurNoire);
            //On ecrit également dans l'image actualisée
            //pour qu'à chaque déplacement d'une enveloppe le texte reste
                SDL_BlitSurface(texte,NULL,image_actualise,&positionInfo);
                SDL_BlitSurface(texte,NULL,ecran,&positionInfo);
                break;
```

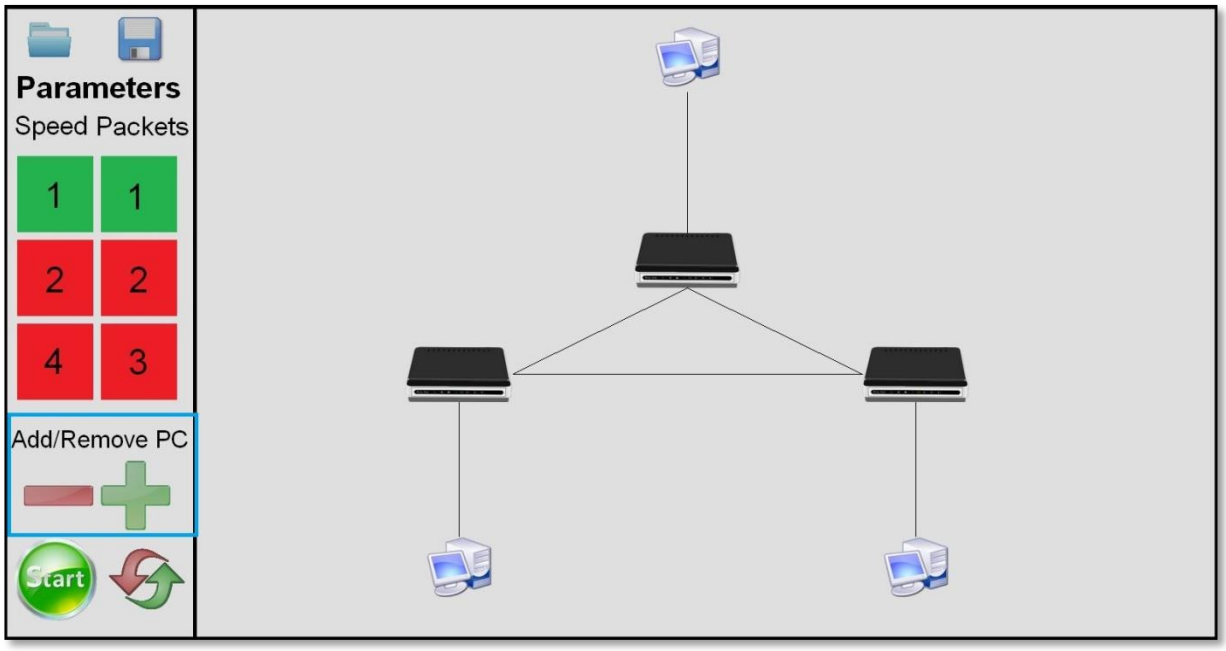
Nous avons ainsi détaillé toute la conception de notre programme. Voyons maintenant, comment fonctionne notre logiciel du point de vue utilisateur.

III. Fonctionnement du simulateur

L'utilisation du logiciel se veut très intuitive. Nous allons expliquer dans cette partie comment utiliser le simulateur et effectuer les réglages.

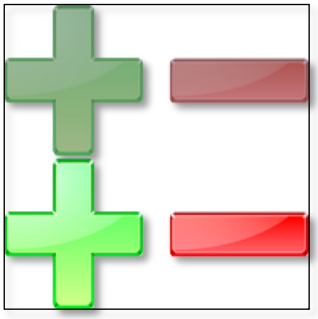
a) Ouverture du logiciel

A l'ouverture du logiciel, la fenêtre suivante s'affiche :



Un réseau de base est déjà créé. Il est constitué de trois nœuds et de trois ordinateurs. Chaque PC est connecté à un nœud différent. Cela permet de simuler l'envoi de paquets à travers un réseau de base.

Pour faire évoluer le réseau, on peut ajouter ou supprimer des ordinateurs grâce aux boutons « + » et «-», qui passent en surbrillance lorsqu'on clique dessus.



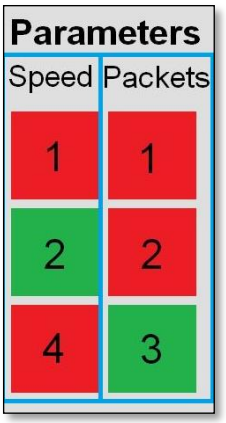
Pour l'ajout d'ordinateurs, il faut cliquer près d'un nœud pour placer le PC. Ce dernier se connecte automatiquement au nœud le plus proche. Des zones ont été prédéfinies pour chaque nœud. Cependant, la position du curseur lorsqu'on place un ordinateur influencera les résultats. En effet, le PC sera plus ou moins proche du nœud en fonction de l'endroit choisi, ce qui influera sur la longueur des fils et donc du temps de transmission.

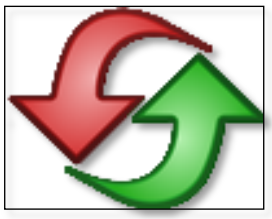
Pour supprimer un PC, il suffit simplement de choisir l'ordinateur en question, en ayant préalablement choisi la fonction de suppression. Il faut ensuite penser à désélectionner les boutons précédemment évoqués pour passer à l'étape suivante.

b) Simulation

Une fois le réseau construit, l'utilisateur sélectionne alors les paramètres du réseau.

Il a le choix entre le nombre de paquets à envoyer (1 à 3) et la vitesse d'émission des paquets (1, 2 ou 4). Cette vitesse est à considérer comme un indicateur : en choisissant « 2 », le message se déplacera deux fois plus vite qu'avec le paramètre « 1 ».





Il sélectionne ensuite l'ordinateur émetteur puis l'ordinateur récepteur. Il peut aussi bien procéder à leur désélection en cliquant sur le bouton « reboot ».

Lorsque tous ces réglages sont effectués, l'utilisateur peut alors procéder à la simulation en cliquant sur le bouton « run ». Lors de la simulation, il suit en temps réel le déplacement du message et ses trajectoires. Il est aussi possible d'arrêter la simulation à tout instant en cliquant sur le bouton rouge « cancel ».

c) Fin de simulation

A la fin de la simulation, les résultats s'affichent à l'écran :
- le nombre de paquets reçus ;
- le temps de transmission ;
- un message qui confirme la bonne réception de tous les paquets émis.

```
Packets reception : 1..2..3
Time : 17.560s
OK
```

Le bouton « run » procède à l'arrêt de la simulation précédente. L'utilisateur a la possibilité de changer les paramètres du réseau.

En lançant une nouvelle simulation, il obtient des résultats différents de ceux obtenus auparavant, dû au chemin aléatoire entre chaque nœud.

La sauvegarde du réseau qu'il vient de construire s'effectue en cliquant sur l'icône de la disquette. Les données sont sauvegardées dans un fichier binaire qui contient le nombre de PC, leur position et le nœud auquel ils sont connectés.

Il est alors possible de réutiliser ce réseau en l'ouvrant par un simple clic sur l'icône du dossier.

IV. Bilan

1. Difficultés rencontrées

a) Commencer le projet

Notre première difficulté a été de démarrer le projet. Nous avions une idée générale de ce qui nous était demandé, mais nous ne savions pas comment nous y prendre. En ce sens, cela n'a pas été évident de se répartir les tâches.

b) Répartir les tâches

Nous avons d'abord travaillé de la façon suivante : Valentin, Thibault et Domitille ont commencé par créer des fonctions et des structures comme enseigné au premier semestre en Langage C. Pendant ce temps, Antoine effectuait des recherches afin de se familiariser avec les bibliothèques graphiques.

Nous avons perdu du temps car nous nous sommes rendu compte que l'on ne pouvait fusionner la programmation sur console avec l'interface graphique. De plus, la bibliothèque GTK s'est avérée difficile à prendre en main. Nous avons donc opté pour SDL, plus adaptée pour la gestion des images.

La répartition des postes a été modifiée. Domitille et Valentin se sont alors consacrés à l'ouverture du logiciel et à sa fermeture. Antoine s'est chargé de l'initialisation du logiciel (éléments fixes, boutons et réseau de base). Une fois fini, il a, avec Thibault, travaillé sur le paramétrage du réseau et sur la gestion des événements.

2. Evolutions possibles du logiciel

Nous avons pensé à quelques améliorations possibles de notre logiciel, la liste est bien sûr non exhaustive.

a) Modification des éléments du réseau

Pour l'instant, il est seulement possible de modifier le nombre d'ordinateurs. L'idéal serait de pouvoir ajouter ou supprimer des nœuds. L'utilisateur pourrait alors gérer des réseaux plus complexes.

b) Envoyer plusieurs paquets simultanément

Lorsque l'utilisateur veut envoyer trois paquets, il ne peut envoyer qu'un seul message à la fois. L'objectif serait de pouvoir envoyer plusieurs messages simultanément avec des émetteurs et récepteurs différents. Cela mettrait en évidence la gestion des conflits.

c) Modifier les paramètres des fils individuellement

Afin que la simulation soit la plus réelle possible, il reste un point à améliorer : le paramétrage des fils. Pour l'instant, il est seulement possible de modifier l'indicateur de vitesse global. Pouvoir modifier la longueur et le débit de chaque fil indépendamment les uns des autres présenterait un intérêt certain.

3. Conclusion

a) Ce qui a fonctionné

Nous avons réalisé une interface graphique avec un réseau de base et un paramétrage possible des éléments. L'envoi de messages d'un émetteur vers un destinataire s'effectue sans encombre. Le message de fin de simulation nous permet d'analyser le temps de transmission de données en fonction des paramètres définis.

b) Ce que ce projet nous a apporté

Lors de ce projet, nous avons apprécié le fait d'être autonomes et libres dans nos choix. Bien que cela nous ait ralentis au début, cette liberté nous a permis de nous écarter légèrement du cahier des charges afin de contourner les problèmes rencontrés.

Comme prévu, cette expérience a été l'application directe des notions apprises lors du module Langage C du premier semestre. Nous avons aussi appris à créer et gérer une interface graphique grâce à la bibliothèque SDL.

Enfin, nous avons découvert des notions sur l'organisation des réseaux, ce qui constitue pour nous un préambule au cours de l'année prochaine.

INSA de Rennes
20 Avenue des Buttes de Coësmes
CS 70839
35708 Rennes Cedex 7

Tél. +33 (0) 2 23 23 82 00
Fax +33 (0) 2 23 23 83 96

www.insa-rennes.fr

INSA

UNIVERSITÉ EUROPÉENNE DE BRETAGNE

Cti Commission des Titres d'Ingénieur

European Accreditation of Engineering Programmes EUR-ACE®

MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE