Simulation de TCP/IP : Rapport de projet

Projet IN608

CAMBRESY Florian
CHALAUD Jean-Christophe
DOS SANTOS Gabriel
GRUCHET Quentin
LE DENMAT Mickaël
LIN Raphaël
MECHRI Fadi
RAMANANDRAITSIORY Johann

TCP/IP Simulation

Université de Versailles Saint-Quentin-en-Yvelines

Mai 2021

Table des matières

1 Introduction

Ce document présente notre rapport final pour l'application qui a été développée dans le cadre du projet "Simulation TCP/IP". Ce dernier est réalisé pour le module IN608 de L3 Informatique, au sein de l'Université de Versailles - Saint-Quentin-en-Yvelines et est encadré par Mme Leila Kloul. L'objectif de ce projet a été de créer une application à but pédagogique simulant le protocole TCP/IP.
Pour cela, l'application doit présenter l'encapsulation d'une donnée afin de circuler sur un réseau, la circulation de cette donnée et sa désencapsulation en vue d'un traitement. Elle devra en outre montrer les différentes étapes du contrôle de congestion TCP/IP. L'application est supportée sur n'importe quel système d'exploitation basé sur le noyau Linux et est compilée simplement et automatiquement grâce à l'outil CMake.

Nous évoquerons dans ce compte rendu toute les étapes qui vous aideront à comprendre le fonctionnement de notre application ainsi que les problèmes rencontrés tout le long de la programmation concernant les cinq modules qui composent notre projet.
En parcourant ce document vous connaîtrez les différents contributeurs ainsi que leur participation au sein du projet. Ce compte rendu sera bien évidement accompagné des fichiers du code source et des entêtes. Vous constaterez la présence des figures deux et trois qui présentent l'arborescence de nos différents fichiers, autrement appelée, la hiérarchie du projet. Enfin nous vous montrerons les aspects techniques inhérents au travail effectué.

2 Explication de l'architecture

2.1 Organigramme et découpe en module

Voici un rappel de l'organigramme prévu dans le cahier des charges :

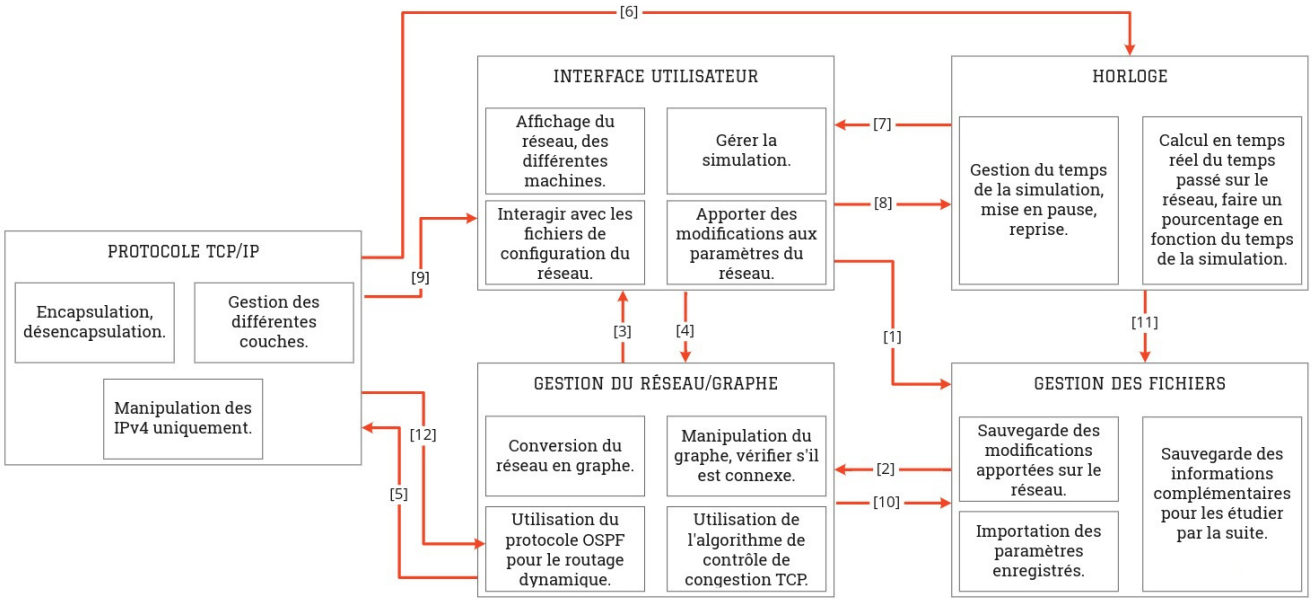

FIGURE 1 – Organigramme des modules avec leurs interactions

[1] : Envoi des signaux pour la création, le chargement et la sauvegarde du réseau (nom, type).
[2] : Importation des fichiers pour exploitation.
[3] : Envoi des informations liées aux réseaux.
[4] : Envoi des paramètres définis par l'utilisateur (ajout de routeur, console).
[5] : Envoi des signaux de lancement du protocole.
[6] : Envoi des information des signaux (démarrage, temps exécution).
[7] : Envoi de l'état des *timers* en temps réel.
[8] : Envoi des informations de modification de l'horloge (pause, lecture, arrêt).
[9] : Envoi des traces d'encapsulation et de désencapsulation.
[10] : Sauvegarde de l'état du réseau.
[11] : Sauvegarde des *timers*.
[12] : Envoi des résultats de l'algorithme.

Voici l'arborescence de notre projet :

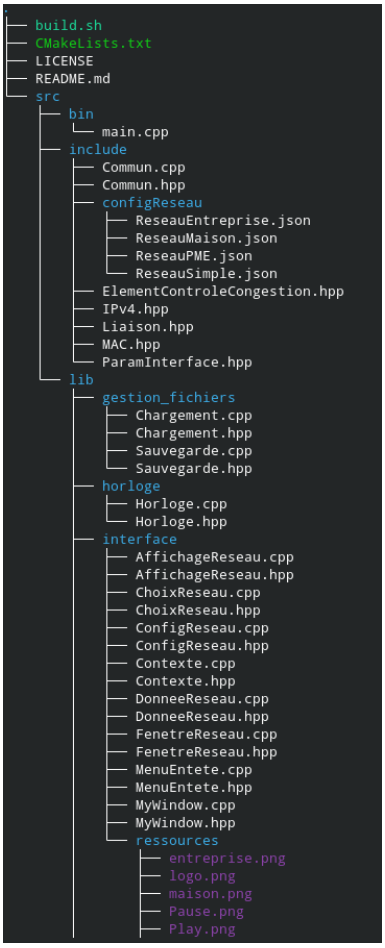

FIGURE 2 – Arborescence de notre programme partie 1

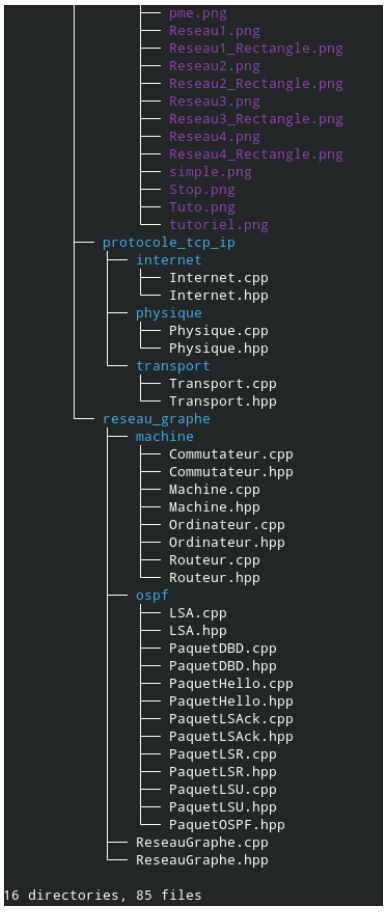

FIGURE 3 – Arborescence de notre programme partie 2

3 Description du fonctionnement de l'application

Lorsque l'application se lance, le réseau simple est chargé par défaut. L'utilisateur peut alors sélectionner une configuration pour la simulation d'un transfert de paquets via le protocole TCP/IP. Il peut choisir les machines de départ et d'arrivée, la taille de la fenêtre, le nombre de paquets et le type des paquets. Il doit ensuite appuyer sur le bouton de validation pour affecter ces caractéristiques au réseau.
Dès lors, le programme lance le routage dynamique afin de créer les liaisons entre les routeurs du réseau. Les fonctions du protocole OSPF se lancent et permettent l'échange d'informations entre les routeurs, leur permettant ainsi d'avoir une connaissance de l'ensemble du réseau et de permettre l'envoi de paquets entre deux machines distantes. Ensuite, tout le noyau de notre application se met en route. Le contrôle de congestion qui permettra l'envoi des fichiers se lance. Ce système appelle lui même plusieurs modules du projet : l'Horloge afin de chronométrer les temps de transport des paquets, le Protocole TCP/IP afin d'encapsuler et désencapsuler les données dès que nécessaire. Elle permet en plus le remplissage de tableau dont les fonctions sont d'enregistrer les temps de parcours des données, les modes du contrôle de congestion (slow start, congestion avoidance, fast retransmit et fast recovery) et la valeur du Cwnd.
L'utilisateur n'a plus qu'à appuyer sur le bouton pour lancer la lecture afin d'afficher toutes les informations obtenues.
Bien évidemment, tout ce processus exécuté en arrière-plan est caché à l'utilisateur afin de garantir une clarté de lecture des informations. En fonction du nombre de paquets envoyés et de la valeur de la taille de la fenêtre, ce processus peut s'avérer plus ou moins long. En effet, si l'on s'arrange pour que le contrôle de congestion ne reste qu'en slow start, le programme s'avèrera extrêmement rapide. A contrario, plus le programme passe du temps dans le mode congestion avoidance, plus le programme mettra du temps à afficher les informations et le graphe dans l'application.
En plus de tout cela, l'utilisateur a la possibilité de sauvegarder les résultats obtenus et de charger manuellement un réseau.
Nous ne détaillerons pas ici comment utiliser notre application. En effet, elle dispose d'un bouton permettant d'ouvrir un tutoriel très explicite. L'utilisateur peut à tout moment actionner le bouton en forme de point d'interrogation pour l'ouvrir.

4 Description des points délicats

4.1 Interface utilisateur

En ce qui concerne l'implémentation de l'interface graphique, certains aspects se sont avérés plus complexes à réaliser que d'autres. La modélisation du graphique fut délicate à mettre en place. En effet, au vue des restrictions imposées par la bibliothèque Qt, il a été difficile de parvenir à nos attentes, notamment à représenter une seule courbe composée de plusieurs couleurs. La mise en commun des différents modules au sein de l'interface graphique était la plus redoutée pour les membres du groupe. Néanmoins grâce à cette fusion, l'interface graphique nous a permis d'éviter tout risque d'erreurs et d'en repérer lorsqu'il y en avait dans les module cibles en question.

4.2 Gestion des fichiers

L'implémentation de la gestion des fichiers fût longue car il fallait écrire à la main les fichiers de configuration JSON, mais fût aussi plus complexe que ce à quoi nous nous attendions. En effet, l'utilisation d'une bibliothèque externe nous a grandement aidé mais il nous a fallu apprendre à l'utiliser et à manipuler des objets JSON, un langage de balisage qui nous était inconnu.

4.3 Gestion du réseau/graphe

Ce module est découpé en trois parties :

4.3.1 Les machines

Pour l'implémentation des machines, nous nous sommes confrontés à quelques erreurs de réflexion. Notre première idée était de partir sur des files d'attentes et de simuler l'envoi par le fait d'empiler un élément dans la file d'attente du voisin et d'appeler la méthode recevoir du voisin qui va dépiler et ainsi de suite. C'est une bonne idée mais nous nous sommes aperçu que l'envoi et la réception n'était pas identique dans le sens émetteur - récepteur et inversement, nous avons donc été obligés de modifier le cahier des spécifications, en modifiant le types ou/et le(s) paramètre(s) afin de pallier ce problème. De plus les méthodes pour le contrôle de congestion nous ont également donné du fil à retordre. Il a malheureusement fallut changer quelques petits détails afin de parvenir à nos objectifs initiaux.

4.3.2 Le protocole de routage OSPF

L'implémentation du protocole de routage dynamique n'a pas posé de problèmes majeurs. Afin d'optimiser les performances, un certain nombre d'attributs des classes des paquets OSPF sont déclarés avec des pointeurs plutôt qu'avec des objets statiques, ce qui permet de réduire les copies mémoires.

4.3.3 La classe ReseauGraphe

La classe ReseauGraphe nous a posé quelques soucis lors de l'implémentation puisqu'elle est centrale au projet. Afin d'avoir accès à certaines informations sur les machines que la classe contient, nous avons rendu certaines méthodes statiques, ce qui nous permet de les appeler sans instance de la classe. Mis à part ces petites modifications, le code a été simplifié et rendu plus facile à lire en découpant certaines méthodes en plusieurs plus petites "sous-fonctions", ce qui offre une meilleure lisibilité et clarté du code source. Malheureusement, la méthode `estConnexe` n'a pas été implémentée car cette méthode n'est pas nécessaire dans notre projet sachant que

nous avons configuré les réseaux nous-même et nous savons qu'ils sont connexes. Ceci n'empêche pas le bon fonctionnement de l'application mais cette méthode serait nécessaire si nous n'avions pas implémenté nous-même les réseaux.

4.4 Horloge

Il n'y a pas eu de problème lors de l'implémentation du module Horloge, il a été un peu plus long que prévu suite aux réflexions pour les tests de ce module.

4.5 Protocole TCP/IP

Pour ce module, nous avons dû revenir à plusieurs reprises sur les fonctions d'encapsulation et de désencapsulation. En effet, elles nécessitent une concaténation ou une division des *bitset* et il est facile de rapidement se perdre. De plus, les fonctions ont été remaniées de nombreuses fois de sorte à ce qu'elles collent au besoin du module ReseauGraphe.

5 Aspect technique

5.1 Fonctionnalités fonctionnelles/non fonctionnelles

5.1.1 Interface utilisateur

En ce qui concerne l'interface utilisateur toutes les fonctionnalités attendues sont opérationnelles. Cependant, la fenêtre de dialogue ajoutée lors du chargement de la simulation après avoir appuyé sur le bouton valider n'apparaît pas pour tous les utilisateurs. C'est un problème dépendant de la machine de chacun qui n'est pas résolu.

5.1.2 Gestion des fichiers

Concernant le module Gestion des fichiers, tout ce qui a été prévu dans le cahier des spécifications a été implémenté, c'est à dire la sauvegarde de configuration, et le chargement d'un réseau.

5.1.3 Gestion du réseau/graphe

Dans le module gestion du réseau/graphe, le routage dynamique est pleinement implémenté. La création de toute les machines est aussi faite. Plus précisément l'envoi et la réception des paquets marche dans les deux sens et enfin dans le contrôle de congestion, les algorithmes de *Slow Start*, *Congestion avoidance*, l'accusé de réception des paquets, la vérification des paquets perdu sont mis en place.
Ce qui ne fonctionne pas se situe dans la partie machine, le renvoi des paquets perdus, cela se répercute donc sur l'algorithme de *Fast retransmit* et *Fast recovery* mais aussi sur la gestion du temps de vie de paquet, la perte des paquets, la synchronisation, la fin de session et le refus des paquets avec un mauvais *cheksum*.

5.1.4 Horloge

Tout ce qui a été prévu dans le cahier des spécifications, avoir une horloge pour calculer de temps mis par les paquets.

5.1.5 Protocole TCP/IP

Pour le module TCP/IP toutes les fonctionnalités prévues ont pu être implémentées et sont fonctionnelles.

5.2 Potentielle(s) amélioration(s)

L'application étant à but pédagogique nous avons décidé de privilégier la visualisation du contrôle de congestion au détriment de l'efficacité de l'application, nous pourrions donc améliorer cette aspect en changeant le système d'envoi et de réception des paquets avec des tubes de communication ainsi que des *threads*.

6 Organisation

Pour l'organisation interne, le travail a été réparti selon le découpage annoncé dans le cahier des spécifications à quelques exceptions près que nous évoquerons durant la présentation.

Pour la réussite de notre projet, celui-ci a été fractionné en cinq modules, avec une équipe affectée par module. Chaque personne a été assignée à une fonctionnalité pour mener à bien ses objectifs. Vous trouverez ci-contre les différents acteurs ainsi que leurs modules et fonctionnalités respectives :

Personnes	Modules	Fonctionnalités	Temps estimé	Lignes de codes
Florian	Gestion réseau graphe et horloge	Conversion réseau graphe et timer simulation	25h	250
Gabriel	Gestion réseau graphe	Protocole OSPF	30h	500
Mickael	Gestion réseau graphe et Gestion fichiers	Contrôle de congestion et sauvegarde	30h	550
Raphael	Gestion fichiers et Interface utilisateur	Chargement et interaction fichier/réseau	25h	250
Johann	Interface utilisareur	Affichage réseau et Gérer la simulation	25h	250
Jean-Christophe	Interface utilisareur	Apporter modification au paramètre réseau	25h	250
Quentin	Protocole TCP/IP	Encapsulation, manipulation ipv4 et gestion des couches	25h	250
Fadi	Protocole TCP/IP	Desencapsulation, manipulation ipv4 et gestion des couches	25h	250

6.1 Organisation globale

L'application est composée de deux parties majeures : la partie graphique et la partie simulation. Ainsi le groupe s'est naturellement découpé en deux, avec d'un côté Gabriel, Mickaël, Florian, Quentin et Fadi et de l'autre Raphaël, Johann, et Jean-Christophe. Les membres des deux groupes ont donc travaillé conjointement afin de s'assurer de la bonne continuité et de la bonne intégration de chaque fonctionnalité les unes par rapport aux autres. Pour l'unification du travail de chaque personne, un dépôt GitHub a été créé par Gabriel. Au début de l'implémentation chaque personne a dû *fork* le projet afin de posséder une copie dans son GitHub personnel

pour qu'elle puisse travailler dessus. Une fois le travail fini et testé, elle n'a plus qu'à faire une *Pull Request* (PR) et que Gabriel valide ou non la fusion des dépôts.

6.2 Organisation individuelle

Chaque personne a complété un fichier *Google Calc* faisant office de planning qui a déjà été envoyé a la personne qui encadre ce projet. Chaque membre du groupe va désormais expliquer comment il s'est organisé, va présenter son temps de travail et son nombre de lignes de code.

6.2.1 Gabriel

J'étais chargé de l'implémentation de l'intégralité du protocole OSPF, ainsi que de la construction des tables de routage des routeurs du réseau. Je n'ai pas eu de difficultés particulières à implémenter un premier jet du protocole OSPF et de l'algorithme de plus court chemin Dijkstra. Cependant, j'ai passé beaucoup de temps à en optimiser les performances, et ce afin que le temps passé dans cette partie de la simulation soit minime et n'affecte pas l'expérience de l'utilisateur. Après avoir fini ma partie, j'ai également aidé mon camarade Mickaël pour l'envoi des paquets au travers des routeurs. Enfin, j'ai participé aux nombreux tests réalisés pour trouver d'éventuels bogues dans l'application. J'ai écrit un total d'environ 1300 lignes de code pour 40h de travail. Tout au long de ce projet, j'ai apprécié tenir le rôle de coordinateur du dépôt GitHub que m'ont attribué mes camarades. Cela demandait parfois beaucoup de travail, notamment afin de m'assurer que le code source respectait bien les règles de syntaxe imposées par le cahier des spécifications, ou bien de corriger les conflits apparaissant parfois lors de l'intégration de nouvelles fonctionnalités au projet. Cependant, ceci m'a permis de garder une vue d'ensemble sur le projet, sur toute la durée de celui-ci.

6.2.2 Mickael

J'étais en charge d'une partie du module Gestion des fichiers, la sauvegarde, et d'une partie du module de Gestion du réseau/graphe, la création des machines sur le réseau, le contrôle de congestion, l'envoi et la réception des trames sur le réseau. La partie création des machines était courte puisqu'il me suffisait simplement de recopier ce qui avait été établit dans le cahier des spécifications. J'ai par la suite entamé une phase de test qui s'est déroulé sans encombre. Après cela je devais m'occuper de la sauvegarde et pour aider mon camarade Raphaël, et afin de finir complètement ce module qui est utilisé dans l'interface et pour les tests des envois, j'ai fait les fonctions de chargement. Enfin à l'aide de mon camarade Jean-Christophe, qui était en avance, nous avons écrit les fichiers de configuration JSON. J'ai à nouveau lancé une phase de test qui se passait bien. La partie compliquée fut pour l'envoi des trames et le contrôle de congestion du à une mauvaise idée d'implémentation lors du cahier des spécification ce qui m'a empêché de faire tout ce qui était prévu. Pour tous cela j'ai mis 80h et j'ai écrit 1210 lignes pour le module Gestion de fichiers et 1704 lignes de code pour le module Gestion du réseau/graphe sans compter les fichiers 'headers', les fichiers 'tests'.

6.2.3 Florian

En ce qui me concerne, je n'ai pas eu de gros problèmes pour l'implémentation de la classe Horloge, j'ai pris du temps surtout pour vérifier mes tests sur l'horloge. Cependant j'ai eu des problèmes lors de l'implémentation de la classe ReseauGraphe. On a modifié les méthodes de nombreuses fois suite à de mauvaises pistes ce qui m'a mis en retard sur mon planning. J'ai finalement réussi à implémenter ce que je voulais avec l'aide de Mickaël et de Gabriel. En ce qui

concerne le temps et les lignes de code, pour la classe Horloge j'ai fait 190 lignes, test compris en 9h. Pour la classe ReseauGraphe, j'ai fait 171 lignes en 15h. Au total 361 lignes de codes pour 24h sur l'ensemble de ma partie avec une dizaine d'heures en plus pour aider les parties qui en avaient le besoin, pas forcement à implémenter mais à partager de la façon d'implémenter avec les autres parties pour que chaque partie puisse s'imbriquer correctement.

6.2.4 Quentin

Pour ma part, j'ai finis assez rapidement ma partie du protocole TCP/IP. Aucune fonction ne m'a posé de réels problèmes. Cependant, j'ai du revenir à de maintes reprises sur ces fonctions, pour coller à ce que le module Gestion du Réseau/Graphe attendait précisément. De plus, j'ai voulu aller aider les autres équipes afin de combler d'éventuels retards. Voilà pourquoi, j'ai été amené à travailler dans le module Interface Graphique où j'ai aidé à faire les schémas des réseaux. J'ai également travaillé sur le module Gestion du Réseau/Graphe où j'ai aidé notamment pour la documentation et le contrôle de congestion TCP/IP. Enfin j'ai travaillé dans les fichiers JSON du module Gestion des Fichiers, où j'ai fait les liaisons entre les différentes machines des réseaux pré-définis. Pour conclure je suis à 53h de travail avec comme nombre de lignes de code environ 1372, tous modules confondus.

6.2.5 Fadi

J'étais en charge de la partie désencapsulation, et checksum dans le module Protocole TCP/IP. Je n'ai pas rencontré d'extrêmes difficultés tout le long de la programmation mais j'ai quand même été assisté par Quentin pour terminer la désencapsulation sur les couches Internet et Physique, qui ont été refaites à plusieurs reprises. Concernant le checksum, mis à part les problèmes de calcul, après mûre réflexion, j'ai pu finalement terminer les méthodes calculerChecksum et vérifierChecksum. Mon travail sur le module TCP/IP comptabilise un total de 240 lignes de code sur le rendu final pour 23 heures de travail sur ma partie.

6.2.6 Raphael

Je suis parvenu à finir le module Interface Graphique dans les temps, notamment pour les taches qui m'étaient attribuées comme la classe Contexte ou encore la classe MenuEntete qui a eu toutes ses fonctionnalités reliées aux autres classes de l'interface (cela permet par exemple d'avoir un chargement et une sauvegarde de fichiers fonctionnelle, à l'aide du module fichier). Ayant eu besoin du fonctionnement des autres modules pour avancer sur l'interface, j'ai décidé de mettre en oeuvre ce qui était en mon possible pour avertir mes collègues d'éventuels dysfonctionnements ou erreurs, ce qui les a aidé a corriger les problèmes en question en temps voulu. Dès les bugs résolus, je me mettais à travailler sur la classe Contexte qui fait le lien entre l'interface et les autres modules. Mickael m'a proposé son aide pour le module chargement de fichiers, ce à quoi j'ai accepté, du au fait que cela me permettait d'avancer plus vite sur l'interface graphique. J'ai apporté mon aide à Jean-Christophe et Johann pour leurs parties de l'interface, et nous communiquions pour nous entendre sur la bonne mise en relation de nos fonctions. J'ai effectué le design de l'interface ainsi que la création des images nécessaires, puis Quentin est venu m'aider pour les schémas des réseaux. J'ai aussi contribué à la mise en commun du programme, notamment avec Gabriel, avec qui nous avons crée les fichiers nécessaires comme par exemple CMakeLists.txt ou encore build.sh. Je suis donc à 842 lignes de code pour un travail de 59 heures.

6.2.7 Johann

J'ai contribué au module Interface Utilisateur où j'avais plusieurs fonctionnalités à coder. Elles se composent de l'affichage de la configuration du réseau sélectionné, de l'affichage détaillé des informations concernant les différentes machines du réseau (Adresse IP, Adresse MAC, masque réseau), ainsi que du débit entre elles et la visualisation d'un graphique permettant de suivre la simulation. J'ai réussi à respecter mon planning ainsi que la charge de travail qu'y était attribué. Cependant, la première semaine s'est retrouvée malencontreusement compliquée pour moi car le chargeur de mon ordinateur s'est retrouvé dans un état défectueux. C'est pourquoi, j'ai pu compter sur l'aide de mon camarade Raphaël qui a su m'épauler à ce moment là. En ce qui concerne la partie affichage du réseau suivi des informations et éléments qui le composent je n'ai pas eu de réels problèmes lors de l'implémentation. Toutefois, un point que je peux souligner est que la programmation du graphique s'est avérée plus complexe que ce que je m'étais imaginé. En effet, Qt ne permet pas de tracer une seule courbe accompagnée de plusieurs couleurs. Pour contourner cela et répondre aux attentes, j'ai donc du rajouter d'autres trajectoires de courbe que j'ai finalement relié pour n'en former qu'une seule. En conclusion, j'ai atteint un nombre d'heures de travail équivalent à 43h pour un total de 550 lignes de codes en comprenant la documentation.

6.2.8 Jean-Christophe

J'étais en charge dans le module "Interface utilisateur" des classes ChoixReseau qui permet le choix des différents paramètres du réseau et ConfigReseau qui affiche le mode du contrôle de congestion ainsi que le temps de l'envoi des paquets. Durant les différentes semaines de travail, j'ai respecté mon planning et le travail qui m'a été attribué. Raphaël m'a aidé sur les différentes parties où j'avais besoin d'appeler la classe Contexte. Je n'ai pas réellement eu de problème au niveau des parties que je devais implémenter. J'ai apporté mon aide à Mickaël pour le module chargement de fichier, j'ai réalisé les calculs des différentes adresse IP. Je me suis aussi occupé de l'écriture d'une partie des fichiers JSON. Je suis à 38h de travail et j'ai écrit 469 lignes de code pour la partie interface utilisateur et 331 lignes pour la partie Gestion de fichiers.