Université de Versailles Saint-Quentin-en-Yvelines

UFR des sciences

Département d'Informatique

Compte Rendu Final

20 Mai 2020

Simulation de TCP/IP

Encadré par

Leila KLOUL

Réalisé par

Toufik GUENANE, Ghiles BABOU,
Nadjib RAHMANI, Mohamed SAMBA DIALLO,
Akli HAMITOUCHE, Mohamed Ramdane DEBIANE,
Koussaila ARAB, Mahdi LARBI

2019/2020

Table des matières

1 Introduction

TCP/IP est né de la réflexion de chercheurs américains suite à un problème posé par le département de la défense (DoD), cette dernière disposait de plusieurs bases militaires sur le territoire, et chacune de ces bases disposait de sa propre logistique informatique. Cependant ces différentes bases peuvent juste partager leurs informations locales et ne peuvent communiquer entre elles. Pour les relier, il a fallu trouver un système permettant de le faire. De ce fait le Protocole TCP/IP est né.

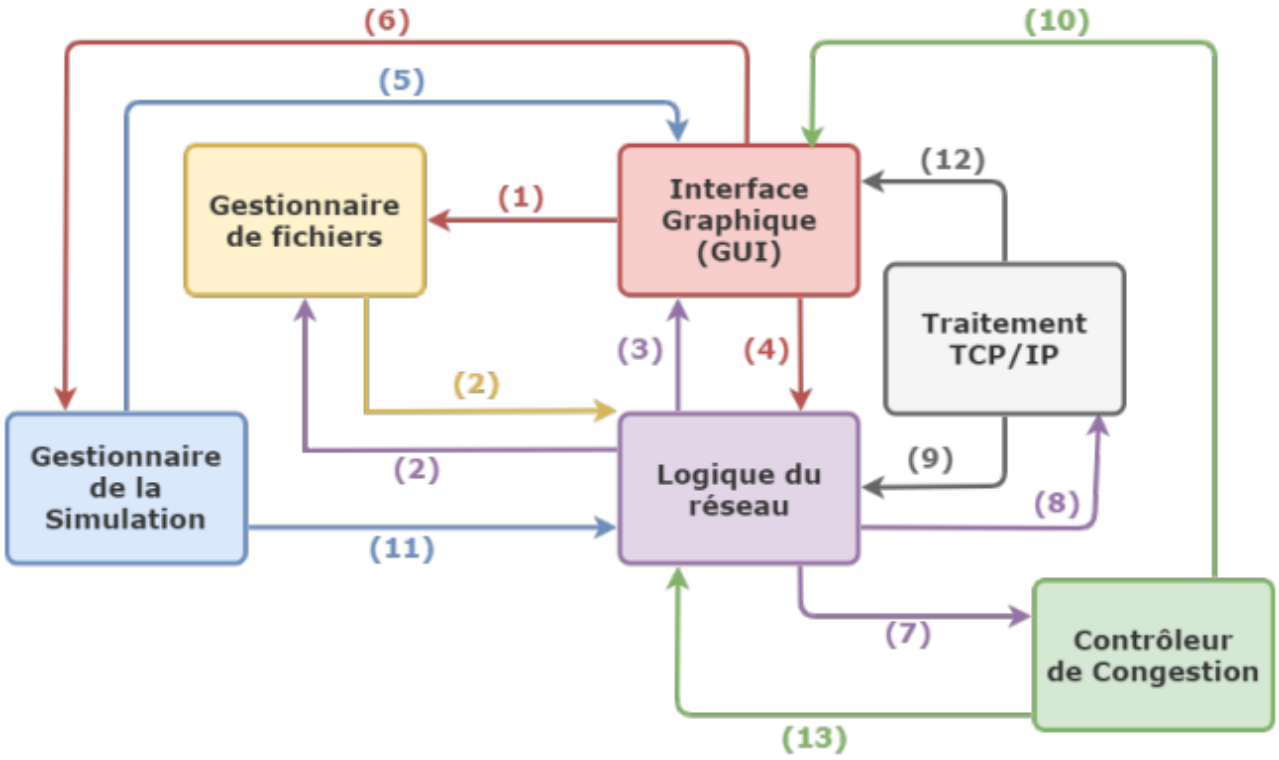
TCP/IP pour Transmission Contrôle Protocol / Internet Protocol, est une suite de protocoles constituant l'ensemble des règles permettant la transmission de données sur un réseau. TCP prend en charge l'ouverture et le contrôle de la liaison entre des ordinateurs et le protocole d'adressage IP assure quant à lui le routage des données transmises ou reçues.

Après la rédaction du cahier des charges et l'analyse des différentes fonctionnalités énoncées dans le cahier des spécifications, nous avons pu implémenter notre Simulateur TCP/IP en respectant les besoins attendus par les clients et les parties prenantes.

Notre simulateur TCP/IP permettra de façon simplifiée la création, la simulation et la visualisation d'un protocole TCP/IP via une interface graphique intuitive. De plus ce dernier a pour but de servir d'outil pédagogique et de support à l'enseignement et à la compréhension des réseaux qui utilisent le protocole TCP/IP.

En plus de l'implémentation, ce document présentera aussi l'architecture et le fonctionnement globale de notre application, les points critiques rencontrés lors de la programmation, les améliorations qui peuvent être apportées à notre projet et enfin l'organisation interne de notre groupe.

2 Architecture de l'application



Après l'étude et l'analyse approfondie du protocole TCP /IP, ses différentes couches, ainsi que les besoins et contraintes imposées par le client, nous sommes parvenus à élaborer l'organigramme ci-dessus qui présente l'architecture de l'application et qui est composé de 6 modules, correspondant chacun à un ensemble de fonctionnalités. La coordination de ces modules nous assure le bon fonctionnement de l'application.

Description de l'architecture :

L'utilisateur à le choix de créer un nouveau réseau ou de charger un fichier contenant un réseau existant, le chemin de celui-ci est envoyé vers le module gestionnaire de fichiers (flèche 1) qui s'occupera de charger les données du réseau, de s'assurer de leur validité puis de les convertir en un graphe qui sera manipulé par le module logique du réseau. (flèche 2) La manipulation de ce graphe englobe la création, la suppression, la mise à jour et la recherche d'erreur de configuration, qui sont retournés à l'utilisateur au niveau de l'interface graphique. Une fois le réseau crée, l'utilisateur peux effectuer des modification sur l'interface Graphique, qui ont un impact direct sur la modélisation du réseau et de chacun de ces composants(flèche 4).Une fois le réseau crée et configuré l'utilisateur peux envoyer des signaux (démarrer, pause, arrêter et redémarrer) depuis l'interface graphique vers le Gestionnaire de simulation(flèche 6), et les timers de l'état d'avancement sont envoyé depuis le Gestionnaire de simulation vers l'interface Graphique(flèche 5). En parallèle, le module Logique du réseau envoi les informations sur le réseau et les données qui circulent vers le gestionnaire de Congestion (flèche 7), qui suite a une série de calculs (RTT, Bande Passante, Latence) et une exécution de l'algorithme de New Reno et renvoie à l'utilisateur, les résultats et une trace d'exécution (flèche 10). Le module logique du réseau lui aussi reçoit les nouvelles valeurs de bande passante et de RTT en vue d'actualiser l'état (les attributs) des câbles (flèche 13). Le Gestionnaire de simulation envoie les informations relatives à l'envoi de paquets (Émetteur, Récepteur, Taille et Type de donnée) vers la Logique du réseau, ainsi que des informations sur l'état et la durée d'exécution qui permettrons une synchronisation des "threads" correspondant à la fonction d'envoi de chaque station (flèche 11). Les informations sur l'envoie des données sont transmise depuis le module Logique du réseau vers le module Traitement TCP/IP (flèche 8), elles servirons à l'encapsulation du paquet avec les bonnes information concernant le paquet à transmettre et le destinataire (flèche 9). Les données générés par ces traitement restent accessibles à l'utilisateur, lui permettant d'étudier plus en détail s'il le souhaite la structure d'un paquet et les mécanismes associés (flèche 12).

3 Prérequis

Afin d'assurer le bon fonctionnement de l'application, l'utilisateur dois avoir vu certaines notions fondamentales des réseaux informatiques, pour pouvoir utiliser au mieux les fonctionnalités proposés par le simulateur.
L'utilisateur doit être capable de :

- faire la distinction entre tous les appareils en terme de raccordement et d'utilisation.
- configurer correctement une topologie (adressage IP, passerelles, adresses réseaux, routes par défaut).
- découper une topologies en sous réseaux et les renseigner dans les tables statiques des routeurs.

4 Fonctionnement des modules

Dans cette section, nous allons expliquer le fonctionnement et le rôle de chaque module au sein de notre simulation, ainsi que toutes les fonctionnalité qu'il fourni :

4.1 Interface graphique :

Ce module regroupe les principales fonctionnalités d'interaction entre l'utilisateur et notre produit, de part le fait qu'il gère les signaux reçus à la souris et au clavier et les transmets ensuite au modules concernés en leur spécifiant un traitement particulier. Ce module propose une série de fonctionnalités dédiés à la visualisation des états, des timers et du déroulement général de la simulation. Parmis les 5 widgets qui composent notre interface, l'objet panneauOutils, regroupe une série d'action impactant le déroulement de la simulation et la manipulation des fichiers en entrée et sortie.

Liste des fonctionalités

- Création du réseau depuis l'interface graphique (Drag Drop).
- Configuration des composants du réseau.
- Visualisation du réseau, des configurations effectuées et des erreurs de cohérence.
- Visualisation de la trace de l'algorithme de congestion.
- Visualisation du timer de la simulation.
- Visualisation des débits sur chacun des câbles du réseau et le temps de liaison entre deux nœuds. Déclencher l'envoi d'un paquet.
- Déclencher l'import, l'export et la sauvegarde d'une topologie sous les différents formats.
- Déclencher le démarrage et l'arrêt de la simulation.
- Choisir (ou changer) le mode de simulation (automatique/manuel).
- Visualiser les informations contenues dans la donnée.

4.2 Gestionnaire de fichiers :

Ce module composé uniquement de fonctions procédurales, contient un ensemble de "parsers" de fichiers, et d'algorithmes permettant de vérifier les données lues depuis les trois formats que nous proposant à l'utilisateur dans le cas d'une importation et/ou exportation.

- Le format XML : Servant à structurer notre fichier de configuration (Acessible en lecture et écriture).
- Le format Dot : qui simplifie la manipulation de graphe via les interfaces Graphiz et xDot.

Liste des fonctionalités

- Lecture du contenu d'un fichier de configuration déjà existant.
- Mise à jour du contenu d'un fichier de configuration déjà existant.
- Conversion du contenu d'un fichier.
- Exporter vers un format de fichier.
- Exporter vers un format d'image.

4.3 Logique du réseau

Ce module modélise l'aspect logique du réseau, il défini ainsi les comportement de chaque appareil ainsi que leurs états tout au long de la simulation. Les relations entre ces appareils quand à elles, sont maintenues à jours dans un graphe représenté par une matrice d'adjacence dans la classe graphe. Nous avons appliqué à cet ensemble de données, des traitement permettant de mettre en oeuvre les fonctionnalités suivantes.

Liste des fonctionalités

- Générer une table de routage.
- Ajouter et supprimer des nœuds et des arrêtes dans la matrice du graphe.
- Vérifier la connexité et la cohérence du graphe.
- Envoyer un message entre deux appareils, sous toutes les contraintes imposés par la simulation.

4.4 Gestionnaire de la simulation :

Ce module, qui est initialisé au lancement du programme, ainsi que les objets qui le composent ont pour rôle de veiller au bon déroulement de la simulation, en orchestrant les changements d'état (ex : Arrêter vers Démarrer). En gardant une trace de l'état et de la durée d'exécution, qui une fois transmises au Gestionnaire de threads, servira à la synchronisation des traitement expliqués à la section suivante.

Gestionnaire de thread :

Lors de la phase de programmation, l'implémentation de l'envoi de donnés sur le réseau, nous a posé un certain problème (cf section 5.1). Une fois la solution à ce problème mise en place et sur un accord général des membres de l'équipe, nous avons lancé l'implémentation d'un architecture parallèle permettant un envoi multiple de paquets entre plusieurs appareils. Le rôle du gestionnaire de thread est d'analyser l'état de la simulation et veiller à la synchronisation des tâches effectués et à l'accès concurrent sur les sections critiques du code. La relation entre le gestionnaire de simulation est une relation de composition. Cette implémentation est basée sur la bibliothèque standard du C++11/14 via les bibliothèque :

- <thread>
- <chrono>
- <mutex>

Ces deux objets permettent de réaliser les fonctionnalités suivantes :

Liste des fonctionalités

- Création d'un timers servant à garder une trace de la durée de la simulation.
- Lancer et arrêter le timer .

4.5 Contrôleur de congestion

Ce module traite le problème de congestion en implémentant la classe Congestion, composant toutes les stations, intervenant dans l'envoi et la réception de paquets pour prévenir et réagir à une éventuelle congestion en appliquant l'algorithme de New Reno. D'autres algorithme, correspondant à des fonctions procédurales, rentrent en jeu dans lors du déroulement de la simulation entre autre dans le calcul des temps de latences, bande passante et RTT.

Ces algorithme sont un support au fonctionnalités suivantes :

Liste des fonctionalités

- Déterminer et gérer une congestion (Algorithme de New Reno).
- Calculer les temps de latence.(Algorithme de calcul de latence dynamique).
- Calculer les RTT (Algorithme de Karn).

4.6 Traitement TCP/IP :

Ce module s'occupe de simuler le fonctionnement de chaque couche de TCP/IP, est attribué à une série de fonctions et procédures permettant d'effectuer les traitements d'encapsulation/Désencapsulation, Fragmentation/Réassemblage

Ce module offre des méthodes utilitaires facilitant l'insertion et l'extraction d'information dans une donnée.

Liste des fonctionnalités

- Encapsulation et désencapsulassions .
- Découpage et réassemblage.
- Création d'un paquet.

5 Les points délicats de la programmation :

5.1 Génération du chemin lors de l'envoie d'un message

La première difficulté rencontrée est celle de la génération du chemin à emprunter par le message lors de la simulation. En effet, lors de l'envoie d'un message d'un émetteur à un récepteur,ce message doit passer par différents équipements et donc le déplacement du message devait être soumis au contraintes inhérentes au réseaux physiques. L'idée initiale était de générer un chemin depuis notre matrice d'adjacence, en utilisant un algorithme de Djikstra modifié pour que celui ci retourne la structure adéquate représentant le chemin. Cette idée n'a pas aboutie, car elle consistait en partie à ignorer les contraintes dues au équipements. La solution à laquelle nous avons convenus, était de générer un chemin correct à partir de la configuration des équipements, c'est à dire depuis leurs tables statiques, leurs passerelles et routes par défaut. Cette solution s'avère moins coûteuse en complexité spatiale et temporelle qu'une exécution de notre version de l'algorithme de Djikstra et se trouve aussi être une bonne base pour l'envoi de message.

5.2 Envoi de message

La deuxième difficulté est l'implémentation de la méthode *EnvoyerMessage*, étant une méthode polymorphique les différents formes de cette méthodes ont pris du temps à être implémentés car ces méthodes devait tenir compte , du sens de l'envoi, de la nature de l'envoi et des possibilités de déplacement depuis un noeud particulier .
La solution au problème de l'envoie de messages, n'as pas été longue à trouver mais nous à fait prendre un léger retard et est désormais fonctionnelle.

5.3 Manipulation de la classe Boost : :Dynamic_bitset

Comme nous l'avons expliqué dans notre cahier des spécifications, nous avons fait le choix de modéliser les données d'une façon la plus simplifiée possible par un tableau à taille variable de bits occupant 1 octet en mémoire chacun. Notre choix c'est porté vers la classe *Boost : :Dynamic_ bitset*, qui offrait un ensemble de méthodes de bases satisfaisantes mais rendait la manipulation de la structure de données longue et fastidieuse. Nous avons donc choisis de résoudre ce problème par l'écriture d'une interface représenté par des méthodes de classes, des fonctions et des surcharges d'opérateurs, pour nous simplifier la manipulation.

6 Changements mineurs des spécifications

Fonctions de manipulation

Pour palier au problème de la manipulation des objets de la classe *Boost : :Dynamic_ bitset*, nous avons ajouté des fonctions utilitaires comment expliqué dans la section précédente.
Voici une liste non exhaustive des fonctions contenues dans DataOutils.cc :

```
int lireNumeroAck (Data * data);
int lireLongueurPaquet ( Data * d);
int lireFlagPaquet ( Data * d);
int lireOffsetPaquet ( Data * d);
int lireTTL ( Data * d);
int lireSommePaquet ( Data * d);
int lireNumeroSequence ( Data * d);
int lireIdIp ( Data * d);
int lireFlagSegment ( Data * d);
int lireNumeroAck ( Data * d);
int lireFenetre ( Data * d);
int lireSommeSegment ( Data * d);
```

Génération de chemin

Lors de la spécification, nous avons fait le choix d'utiliser la méthode *void generer_ table_ chemin()* en $O(n^3 log(n))$, pour déterminer le chemin, cette méthode étant appelée à chaque mise à jour de la matrice, la complexité était mauvaise . Cette méthode à été remplacée par :

```
int genererChemin(int src, int n1, int n2 , vector<Cable *> &path, bool allPath)
```

dont la complexité en $O(n^2)$ est inférieur à celle de la solution initiale, de plus elle n'est appelée qu'une seule fois lors de l'envoi de message afin de générer le chemin entre l'équipement émetteur et récepteur.

Découpage de la mise à jour de la matrice

De plus la méthode *void mise_ a_ jour_ de_ matrice(QEvent *event)* ayant pour but de centraliser tout les traitements en lien avec la mise à jour de la matrice. Lors de l'implémentation, on a constaté que cette fonction avait une trop grande complexité, c'est pour cela qu'on a décidé de la séparer en nouvelles fonctions de complexité minimale.Ces fonctions seront appeler indépendamment à chaque création , suppression (constructeurs et destructeurs) de ces équipements.

```
static void ajoutNoeudMatrice(Noeud * n)
```

Méthode qui permet d'ajouter un équipement à la matrice d'adjacence du graphe du réseau lors de l'ajout d'un équipement par l'utilisateur.
Entrée :

— *Noeud * n :* l'équipement à ajouter à la matrice.

```
static void ajoutCableMatrice(Cable * C)
```

Méthode qui permet d'ajouter un câble à la matrice d'adjacence du graphe du réseau lors de l'ajout d'un câble par l'utilisateur.
Entrée :

— *Cable * C :* le câble à ajouter à la matrice .

```
static void supprimerNoeudMatrice(Noeud * n)
```

Méthode qui permet de supprimer un équipement de la matrice d'adjacence du graphe du réseau lors de la suppression d'un équipement par l'utilisateur.
Entrée :

— *Noeud * n :*l'équipement à supprimer de la matrice .

```
static void supprimerCableMatrice(Cable* c)
```

Méthode qui permet de supprimer un câble de la matrice d'adjacence du graphe du réseau lors de la suppression d'un câble par l'utilisateur.
Entrée :

— *Cable * C :* le câble à supprimer de la matrice .

Afin de faciliter la mise à jours de la tables de routage des équipements, nous avons décidé d'ajouter les deux fonctions *supprimerRoute* (pour la suppression d'une route) et *modifierRoute* (pour la modification d'une route).

```
void supprimerRoute(int id)
```

Supprimer la route numéro id dans la table de routage d'un équipement .
Entrée :

— *int id :* l' identifiant de la route à supprimer de la table de routage.

```
void modifierRoute(int id, Route * route)
```

Modifier une route déjà existante dans la table de routage d'un équipement.
Entrée :

— *int :* Identifiant de la route à mettre à jour.

— **route :* La nouvelle route à insérer à l'identifiant précédent.

Réception de messages

Du fait de l'action de contrôleur sur l'envoi et la réception de messages, inclure une méthode membre qui ne gère que l'envoi, ne fait qu'en partie traitement attendu. Nous avons donc inclus dans la hiérarchie de classe des appareils, une méthode virtuelle pure qui décrit symétriquement le comportement à la réception d'un appareil et de son éventuel contrôleur de congestion (class Congestion)

```
virtual void recevoirMessage(int numseq,int numif, destination dest) = 0
```

Méthode qui permet à l'équipement destinataire de recevoir le message.
Entrée :

— *int numseq :* le numéro de séquence du paquet envoyer .

— *int numif :* numéro de l'interface de l'équipement qui reçois le message .

— *destination dest :* l'équipement qui est censé recevoir le message .

Interfaces

Dans la classe InterfaceFE du module logique de réseau, nous nous sommes servis d'expression régulières et des fonctionnalités du module *Boost : :regex*, pour garantir la véracité des adresses entrés par l'utilisateur et lui retourner un affichage d'erreur. De plus la génération des adresse Mac est faite automatiquement suivant ces expressions régulières.

Câbles

Pour le bon fonctionnement des câbles, nous avons décidé de créer les deux fonctions :

```
bool connexionNoeuds(Noeud* N1, int interface1, Noeud* N2, int interface2);
bool acceptCable(Cable* cable, int idInterface);
```

afin d'éviter d'éventuel bugs et assurer le bon fonctionnement sans erreurs des câbles lors de leurs ajouts ou leurs liaisons aux équipements.

Quand à la seconde, elle à pour rôle de vérifier si un équipement peut être lié à un Cable par l'interface idInterface .

Design Pattern : Singleton

L'utilisation de ce design pattern, répond à un problème très particulier de conception, qui est la création et la manipulation d'une unique instance d'objet. Lors de la programmation du module de logique du réseaux, nous avons remarqué qu'une fois la mémoire allouée à l'objet, et jusqu'à sa destruction, l'ensemble des modules interagissaient avec cette instance seulement.
Nous avons décider de modifier le prototype de la classe pour faire en sorte que l'accès à cette objet ne puisse se produire que par la méthode

```
static Graphe* Graphe::get();
```

qui assurait au développeur une sécurité pour la manipulation de cette instance.
Une autre condition à la mise en place de se design pattern et d'interdire l'affectation et la copie en "blocant" l'operator= et le constructeur de copie.

```
Graphe* operator=(const Graphe* g) = delete;
Graphe::Graphe(const Graphe& g) = delete;
```

Cette syntaxe permet de faire savoir au compilateur que l'utilisation de ces methodes n'est pas possible.

7 Comparaison entre l'estimation et l'implémentation :

Module de l'application	Coût en nombre de lignes	Coût en temps	Personnel(s) en charge
Interface Graphique	Estimation : 800 lignes Implémentation : 1000lignes	Estimation : 25heurs Implémentation : 35heurs	Tous les membres du groupe
Logique du réseau	Estimation : 1700 lignes Implémentation : 1300 lignes	Estimation : 45heurs Implémentation : 40heurs	Tous les membres du groupe
Contrôleur de congestion	Estimation : 250 lignes Implémentation : 250 lignes	Estimation : 15heurs Implémentation : 15heurs	Debiane & Rahmani
Gestionnaire de la simulation	Estimation : 250 lignes Implémentation : 100 lignes	Estimation : 12heurs Implémentation : 10heurs	Babou & Arab
Traitement TCP/IP	Estimation : 300 lignes Implémentation : 600 lignes	Estimation : 25heurs Implémentation : 25heurs	Larbi & Babou & Guenane & Arab & Dialo & Hamitouche
Gestionnaire de fichiers	Estimation : 500 lignes Implémentation : 450 lignes	Estimation : 10heurs Implémentation : 15heurs	Tous les memebre du groupe
Le coût total	Estimation : 3800 lignes Implémentation : 3700	Estimation : 132 Implémentation : 140	

On remarque dans certains cas des différences plus ou moins grandes entre les estimations en nombre de lignes ou d'heures de travail et le coût réel de l'implémentation en heures de travail et ce pour des différentes raisons.

Dans le cas du module Gestionnaire de fichier on peut remarquer qu'il y a une différence entre le taux horaire estimé et celui de l'implémentation, ceci peut être expliqué par le fait que, lors de la rédaction du cahier de charges on a décidé que la sauvegarde des fichiers sera seulement en format XML, et lors de l'implémentation on a rajouté la sauvegarde sous format DOT et PNG.
Pour le cas de l'interface Graphique, elle a connu de nombreuses améliorations au fil du processus du développement, ce qui a entraîné une importante augmentation d'heures de travail et par conséquent une augmentation du nombre de lignes de code.
En ce qui concerne les deux modules Logique du réseau et traitement TCP/IP, on peut expliquer l'augmentation du nombre de lignes estimées pour ce dernier et sa réduction pour l'autre par le fait que, lors de l'élaboration du cahier de charge la création du paquet a été prévue d'être implémenter dans le module logique du réseau, mais lors de l'implémentation on a trouvé qu'il vaudrait mieux l'implémenter dans le module Traitement TCP/IP.

8 Conclusion technique :

Le produit final obtenu après les phases du projet, est très proche du résultat attendu en terme de fonctionnalités. Toutes les fonctionnalités décrites dans les section précédentes sont implémentés et fonctionnelles. Cependant, quelques "bugs" peuvent s'être glissés dans le module interface graphique, ces "bugs" seront corrigé entre la remise du projet et la soutenance. Leur résolution sera indiqué lors de la démo.

D'autre part, pendant l'étape de programmation des potentielles idées d'amélioration ont été suggérés et retenues par les membres du groupe et leur implémentation se fera entre la remise et la soutenance en fonction du temps alloué à chaque tâche.
Parmis ces fonctionnalités : l'envoi multiple (Threads) et la visualisation détaillée des paquets est disponible et offre de bon

résultats.
Quand on fonctionnalités qui seront finalisés pour la soutenance nous avons :

— Analyse avancée des paquets et visualisation grâce à l'interface Wireshark (vue en cours)

— Algorithme RED : l'algorithme choisi actuellement pour le traitement et la gestion de congestion offre de bon résultats dans le cas de petits réseau mais se montre efficace seulement dans façon réactive, c'est à dire , une fois que la congestion à déjà eu lieu.
Une amélioration à cette approche est d'ajouter l'algorithme RED qui agit directement sur les files d'attente des routeurs en supprimant un paquet susceptible de déclencher une congestion.

9 Conclusion sur l'organisation interne au sein du projet :

La réalisation de ce projet nous a permis dans un premier temps de gérer le travail en équipe. Cela n'a pas toujours été facile, puisqu'il a fallu faire des mises au point, des réunions, comprendre ce que les autres personnes du groupe faisaient, et arriver à nous mettre d'accord sur les différentes décisions prises durant le traitement des différents aspects du projet. Ce genre de projet en équipe permet d'apprendre à comprendre les choix des autres, faire des critiques constructives, accepter ses erreurs et se remettre en question.
D'autre part, Le projet tutoré nous a également permis de mettre à profit nos connaissances sur un sujet où est réalisé l'une des plus grandes recherches de l'histoire de l'informatique, et ça nous a permis d'apprendre les différents concepts que ce soit dans le domaine des réseaux, de la programmation, de l'algorithmique et de la gestion de projet. Un projet d'une telle envergure apporte beaucoup,par exemple dans la gestion du temps, car le respect de l'échéancier n'est pas toujours évident. Pour veiller au bon respect des échéancier sur les tâches de chacun, de la bonne entente entre les membres et de leurs communication, nous avons élu un chef de projet et ce dès la première semaine. Ce projet a donc été très instructif pour tous les membres du groupe. Cette expérience nous a apprise que la place de la programmation dans un projet logiciel est bien importante qu'on le pensait au départ, et que les étapes de conception sont tout aussi importantes et déterminantes dans la réalisation d'un projet logiciel.