Rapport de projet de programmation impérative

Picard Arthur, Gourio Jules, Mangenot Guilhem, Rousset Eliott

2022-2023

Figure 1 – Routeur

Table des matières

1 Résumé

Ce rapport est destiné à détailler la réalisation de ce projet concernant les tables de routage. Nous y verrons les différents choix qui ont été faits durant la conception, l'organisation du programme ainsi que des explications concernant le fonctionnement du code. Nous y ferons également un bilan résumant l'état du projet et les connaissances acquises durant la réalisation de celui-ci.

2 Introduction

L'objectif de ce projet est d'implanter, évaluer et comparer plusieurs manières de stocker et d'exploiter la table de routage d'un routeur. Pour cela on s'intéresse à la gestion du cache, qu'on implémente dans ce projet sous la forme d'une liste chaînée puis d'un trie. On expérimente également plusieurs politiques de gestion du contenu du cache.

3 Architecture en modules

Ce programme utilise plusieurs structures de données qui ont chacune un fonctionnement différent. Il est ainsi nécessaire de l'organiser en modules afin de faciliter l'exploitation de ces structures mais aussi pour éviter la redondance du code.

Tout d'abord nous avons donc un module nommé Cache (peut-être à renommer Exploitation SDA pour clarifier) qui comprend toutes les opérations de bases pour manipuler ce type de donnée comme par exemple des fonctions pour enregistrer des éléments, pour en supprimer, pour en chercher... mais surtout des sous programmes servant à trouver la meilleure route dans la table, à trier le cache selon les différentes politiques et à en assurer la cohérence ou à mettre à jour les statistiques.

De la même manière, nous avons un module similaire pour les arbres binaires.

Nous utilisons aussi le module Exploitation table routage qui lui est dédié à la conversion de fichiers textes contenant la table et le cache en structures de données exploitables par notre programme.

Enfin, nous avons créé un module Utilitaire qui contient tous les sous programmes utiles pour les deux structures de données.

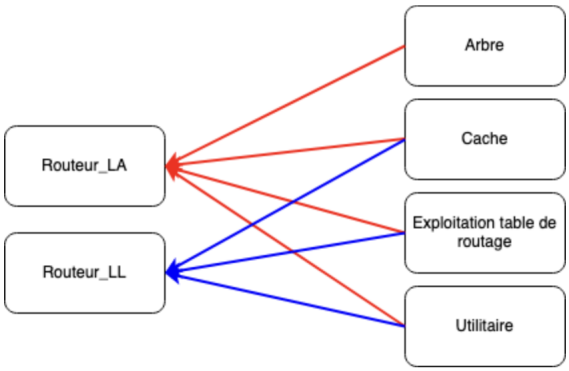


FIGURE 2 – Modules principaux

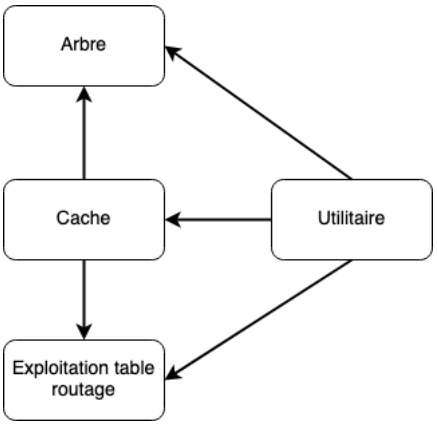


FIGURE 3 – Modules secondaires

4 Principaux choix réalisés

Pour les arbres, nous utilisons une structure hiérarchique (arbre binaire). Nous nous sommes conformés aux exemples du sujets et nous avons donc décidé de n'enregistrer des données que sur les feuilles de l'arbre. Ainsi, les nœuds internes sont des nœuds vides. Nous les reconnaissons en enregistrant "Noeud vide" à la place d'une interface comme "eth0" ou "eth1". Pour retrouver une route dans l'arbre, il suffit de parcourir l'arbre en fonction des bits de la route en binaire, un 0 et l'on va à gauche, un 1 et l'on va à droite. Pour ne pas effectuer trop de calculs, l'enregistrement d'une route ne tient pas compte de tous ses bits mais seulement du premier bit de différence avec les autres routes enregistrées dans l'arbre (le cache) qui possèdent les mêmes premiers bits.

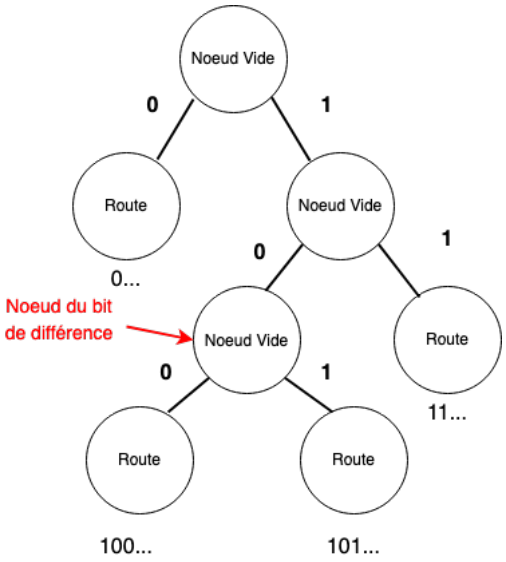


FIGURE 4 – Forme de l'arbre

5 Description des algorithmes

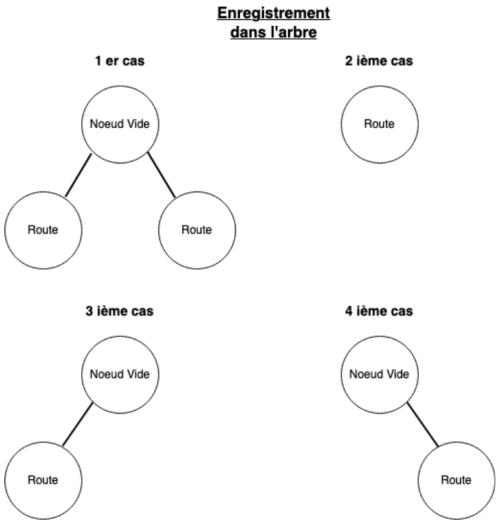
5.0.1 Enregistrer dans l'arbre

Pour enregistrer dans le cache sous forme d'arbre, il est nécessaire de considérer les 4 cas ci-dessus.

— Le premier cas dans lequel les deux fils gauche et droit sont non nuls. On poursuit alors à gauche ou à droite selon l'écriture binaire de la destination.

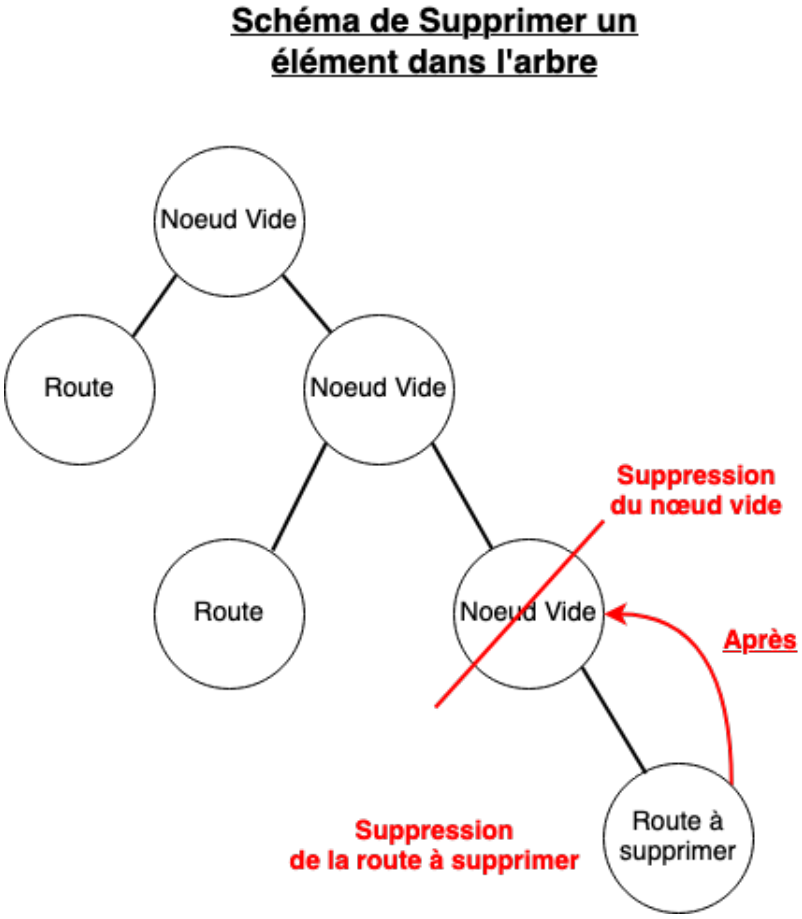
— Le deuxième cas dans lequel les deux fils sont nuls. On rend vide le nœud et on sépare en deux fils gauche et droite au niveau du bit de différence. On passe du cas 2 au cas 1 à la fin de l'enregistrement.

— Le troisième cas où seul le fils droit est nul. Soit on crée une cellule à droite soit on parcourt l'arbre à gauche et on se place alors dans le cas numéro 2.
— Le quatrième cas où seul le fils gauche est nul. Soit on crée une cellule à gauche soit on parcourt l'arbre à droite et on se place alors dans le cas numéro 2.



5.0.2 Supprimer une adresse dans l'arbre

La suppression de l'élément nécessite d'avoir le cache actualisé durant la suppression et l'adresse de l'élément à supprimer. Pour cela, Il est primordial de descendre l'arbre jusqu'à trouver l'élément et le libérer de la mémoire. Cependant, si il y'a des noeuds vides avant cet élément qui ne servent pas il faut les supprimer pour garantir le fonctionnement de enregistrer par la suite.



5.0.3 Trouver meilleure route

Pour l'arbre ou la liste chaînée, on regarde d'abord dans le cache si aucune route correspond à l'adresse que l'on veut router. Si on ne trouve pas de routes alors on part chercher dans la table de routage qui contient toutes les routes.

5.0.4 Tri LRU

Le tri LRU nécessite de retenir les utilisations des routes du cache et de les actualiser à chaque fois qu'un nouveau paquet est traité. Pour cela, à chaque fois

que nous cherchons une route dans le cache, nous enregistrons la dernière fois qu'il a été utilisé. Nous avons donc un compteur qui permet de suivre le nombre de paquets traités. Dès que nous en traitons un nouveau, si la meilleure route pour ce paquet se situe dans le cache nous actualisons la valeur de la dernière utilisation pour cette route. Quand nous devons trier le cache nous supprimons la route dont la dernière utilisation est la plus ancienne.

5.0.5 Tri LFU

Le tri LFU est similaire au LRU à la différence qu'au lieu d'enregistrer la dernière utilisation dans chaque route du cache, nous incrémentons la valeur de la fréquence d'utilisation de cette route à chaque fois qu'elle est utilisée. Dès que l'on en a besoin, nous pouvons donc supprimer la route avec la plus petite fréquence.

5.0.6 Cohérence

La cohérence est le fait de vérifier si les données insérées dans le cache sont valides ou non. Pour cela, avant d'enregistrer dans le cache et une fois que nous avons trouvé la meilleure route pour ce paquet, nous prenons ce paquet et chaque destination de la table de routage et nous les comparons bit à bit. Cette comparaison nous renvoie un masque pour chaque destination qui contient un 1 quand les deux bits correspondants des deux adresses IP sont égaux. Nous augmentons ensuite la longueur, c'est-à-dire le nombre de 1 successifs, de chaque masque créé de 1. Enfin nous gardons le masque le plus long. Nous enregistrons ainsi dans le masque le paquet, le masque trouvé et l'interface trouvée lors de la recherche de la meilleure route.

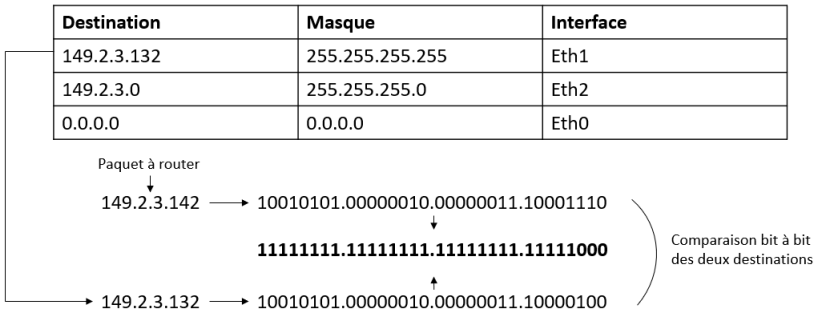


Figure 5 – Fonctionnement de la cohérence

La route enregistrée dans le cache est donc :

149.2.3.136 255.255.255.248 Eth2

6 Types définis et utilisés dans les programmes

Type Politique de tri est énumération (FIFO, LRU, LFU)

Type Arguments est enregistrement (Taille : Entier ; Politique : Politique de tri ; Statistiques : Booléen ; Routage : Chaîne de caractère ; Paquets : Chaîne de caractère ; Resultats : Chaîne de caractère)

Type Adresse IP est mod 2**32

Type LCA est pointeur sur Cellule ;

Type Cellule est enregistrement (Destination : Adresse IP ; Masque : Adresse IP ; eth : Chaîne de caractère ; Dernière Utilisation : Entier ; Nb Utilisation : Entier ; Suivant : LCA)

Type Arbre est pointeur sur Cellule Arbre

Type Cellule Arbre est enregistrement (Destination : Adresse IP ; Masque : Adresse IP ; eth : Chaîne de caractère ; Suivant : Arbre)

Type Donnee est enregistrement (Destination : Adresse IP ; Masque : Adresse IP ; eth : Chaîne de caractère)

7 Démarche adoptée pour tester le programme

Pour tester les différents algorithmes nous avons adopté une démarche de débogage. Dès lors qu'un problème se présentait, nous utilisions différents Put Line() sur les éléments qui posaient problèmes dans chaque algorithme défaillants. Ainsi, après avoir identifié les problèmes, nous les corrigions et retestions de la même façon les algorithmes.

8 Difficultés rencontrées et solutions adoptées

Une des plus grandes difficultés rencontrées a été l'enregistrement dans l'arbre, notre raisonnement logique était correct mais le code était mauvais. Nous utilisions deux arbres dont l'un qui pointait sur l'autre dans une sous-procédure, ce qui avait pour conséquence de créer une copie de ses deux pointeurs, ce qui supprimait le

lien entre eux ainsi en sortie de cette procédure l'un des arbres n'était pas modifié alors qu'il devait l'être. Pour pallier ce problème nous avons décidé de tout coder en récursif ce qui était plus compliqué mais résolvait tout les problèmes de pointeurs.

9 Organisation de l'équipe

Durant toute la durée du projet, nous nous sommes séparées les tâches en deux pour améliorer notre efficacité. Pour le premier rendu, l'équipe constituée de Jules et Arthur a ainsi mis en place le traitement des paquets tandis que Guilhem et Eliott ont réalisé le traitement des fichiers et la prise en compte des arguments lors de l'appel du routeur. Pour la deuxième partie, la première équipe s'est concentrée sur tout le système des arbres et la deuxième a travaillé sur l'exploitation et la cohérence du cache et de ses différents systèmes de tri.

10 Bilan technique

— Une perspective d'évolution du programme pourrait être l'amélioration de la procédure qui supprime un élément dans l'arbre, actuellement celle-ci se contente de supprimer un élément ainsi que les nœuds vides inutiles subsistants. Nous pourrions dans un souci d'optimisation faire une procédure qui remonterait les autres éléments de l'arbre à partir du moment où l'on supprimerait un élément, les autres perdraient leur place optimales.
— Amélioration pour le parcours a gauche parcourir l'adresse à l'envers pour avoir un bit de différence plus faible.
— Possibilité de modifier la table de routage au cours de l'exécution.

11 Bilans personnels

11.1 Rousset Eliott

Ce projet m'a aidé à avoir une meilleure compréhension des concepts de réseau et la façon dont les paquets sont acheminés. La programmation en Ada m'a également aidé en termes de rigueur ce qui sera utile sur tous les langages. Enfin le fait de travailler en groupe sur un projet aussi long était inédit pour moi et était intéressant en termes d'organisation et de séparation des tâches.

11.2 Picard Arthur

Le projet m'a permis de perfectionner mes aptitudes de programmation en Ada. Grâce au travail sur les arbres, j'ai pu bien comprendre les pointeurs et l'utilisation des types. De plus, ce projet à travers les raffinages et la programmation en Ada m'ont permis de développer ma rigueur. Finalement, j'ai trouvé le projet intéressent au niveau de l'organisation.

11.3 Gourio Jules

Ce projet m'a permis d'améliorer mes connaissances en Ada et surtout aux niveaux des tests. Le travail sur les arbres étant très abstrait et difficile à afficher, je suis devenu très compétent dans les tests et les corrections des fonctions. Ce projet à aussi développé ma connaissances des arbres (structure que j'avais découverte en spécialité info). Enfin, vu l'exigence du projet j'ai su développer un fort esprit d'équipe et acquérir une maîtrise de l'organisation.

11.4 Mangenot Guilhem

Ce projet m'a permis de me perfectionner en programmation en langage Ada (notamment concernant le fonctionnement des pointeurs), et l'amplitude du projet m'a appris l'importance de s'organiser et de structurer un code pour qu'il soit flexible et adaptable. Le fait de travailler à plusieurs sur le même projet a pour moi été très enrichissant, autant au niveau organisationnel qu'humain.

11.5 Temps passé

Temps passé sur :	Conception	Implantation	Mise au point	Rapport	Total
Jules Gourio	6h	15h	35h	1h	57h
Arthur Picard	6h	17h	35h	2h	60h
Eliott Rousset	5h	20h	20h	4h	49h
Guilhem Mangenot	5h	18h	24h	1h	48h