Rapport technique

ZEIDLER Mathieu, AUGEREAU Robin et MURUGESAPILLAI Vithursan

Département Sciences du Numérique - Première année 2022-2023

Table des matières

1. Résumé

L'objectif de ce rapport est de donner une vue globale du projet dans sa **conception**, dans la **répartition du travail**, et dans les éventuelles **difficultés rencontrées**. Une présentation générale du projet et du **principe du routeur** permettent de comprendre l'architecture du projet : les **types de données**, les **algorithmes** et leurs **caractéristiques** utilisés. Les **principaux algorithmes** sont présentés dans ce rapport technique avec des **tests** afin de faciliter la compréhension du travail auprès du jury. Un **bilan technique** est proposé, donnant un **état d'avancement du projet** et les **perspectives d'améliorations** du projet. Enfin, chaque membre du groupe soumet un **bilan personnel** faisant part de l'enseignement tiré de ce projet.

2. Introduction et principe du routeur

L'objectif de ce projet est d'**implanter**, **évaluer** et **comparer** plusieurs **manières de stocker** (LCA ou Arbe préfixe) et **d'exploiter la table de routage d'un routeur** afin de mettre en place **un routeur**.

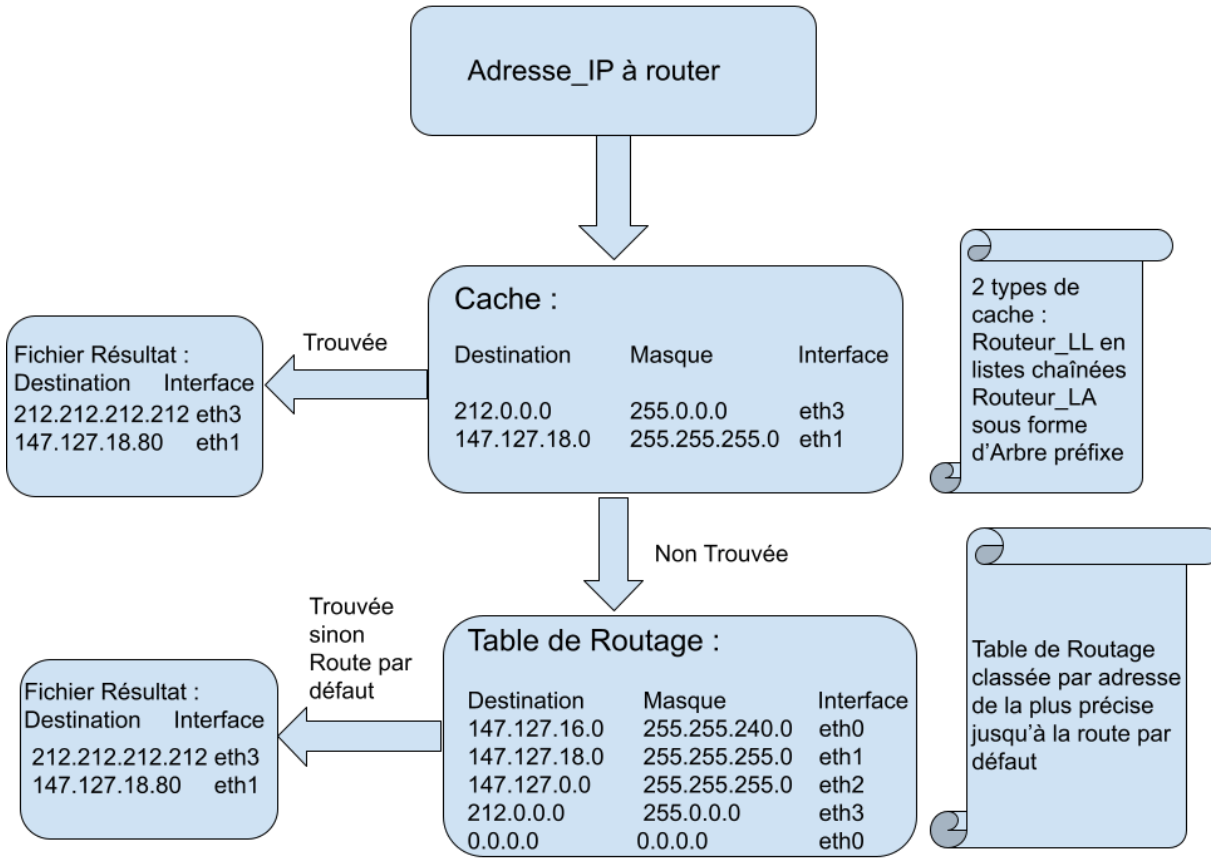
Plusieurs programmes ont donc été rédigés en langage Ada, organisés sous forme de **modules** *(voir Architecture du projet)*. Au fil de la programmation, nous avons débattu et avons abouti à des **choix primordiaux** dans la conception du projet qui sont exposés dans ce document *(voir Principaux choix réalisés)*. Afin de mener à bien ce projet, nous avons dû décider dès le début des **types de données** et des algorithmes principaux que nous allions utiliser *(voir Principaux algorithmes et types de données)*. Dans le cadre d'une conception en programmation défensive, le **test** des programmes écrits est très important puisqu'ils permettent de s'assurer que tous les cas d'utilisation sont couverts *(voir Test du programme)*. Au long de la conception et de l'implémentation de ce projet, nous avons rencontré de nombreuses **difficultés**, qu'il a évidemment fallu **résoudre** *(voir Difficultés rencontrées et solutions apportées)*. Ce projet nécessite l'implantation de plusieurs façons de résoudre le problème demandé, nous distinguerons leurs études et les comparerons *(voir Organisation et répartition du travail)*. Après plus d'un mois de travail, nous avons abouti à un ensemble de programmes fonctionnels dont les **améliorations possibles** sont multiples *(voir Bilan technique)*. En tant que programmeur, nous avons chacun vécu le projet différement, c'est pourquoi nos différentes perceptions sont décrites dans ce document *(voir Bilan personnel et individuel)*.

Routeur wifi

Définition selon le Larousse :

Réseau local hertzien (sans fil) à haut débit destiné aux liaisons d'équipements informatiques dans un cadre domestique ou professionnel

Schéma du fonctionnement du Routeur



3. Architecture du projet

A) Module “TR_Liste”

Ce projet nous fait manipuler une **table de routage** dans laquelle se situent des **adresses IP** de destination, des **masques** ainsi que des **interfaces**. Ainsi, regrouper chacune de ces informations dans des cellules pour créer des **listes chaînées associatives** (LCA), nous permet de stocker des informations efficacement en supprimant et en ajoutant facilement grâce à la manipulation des données dynamiques (pointeurs). C’est pour cette raison que nous utilisons le principe des LCA pour **la table de routage** et **le cache** (routeur_LL).

B) Module “ABR”

Une autre manière de gérer **le cache** est d’utiliser le principe de l’**arbre préfixe**. En effet, comme il est constamment mis à jour en supprimant et ajoutant certaines données, cela peut-être plus efficace d’utiliser un arbre préfixe qui est **plus performant**. Sa meilleure performance provient du fait de stocker des données de façon organisée pour retrouver rapidement un donnée précise en comparant sa clé à chaque nœud de l’arbre.

C) Module “CLI”

Ce module regroupe les **lignes de commande** ainsi que les différentes **opérations sur le type Adresse_IP**. En effet, nous devons réaliser plusieurs **conversions** de type par exemple lorsque nous sélectionnons des adresses IP d’un fichier texte, nous devons ensuite les convertir en type T_Adresse_IP pour réaliser des actions dessus.

4. Principaux choix réalisés

Nous avons choisi d’utiliser des modules génériques avec des paramètres de généricité pour pouvoir manipuler facilement les différents types en les instanciant comme dans ces exemples :

```
package TR_Routeur is new TR (T_IP => T_Adresse_IP, T_Masque =>
T_Adresse_IP, T_Interface => Unbounded_String);
```

package TR_package is new TR (T_IP =>String, T_Masque => String , T_Interface => String);

Devant faire régulièrement la transition entre des chaînes de caractères et des types T_Adresse_IP, nous avons choisi de mettre en place un algorithme assurant cette transition s'appelant "parse". De plus pour passer d'un String à un Unbounded String nous avons également créé une fonction pour les convertir en utilisant "+". Toutes ces fonctions nous ont permis de manipuler plus facilement les types et de limiter les problèmes de compilation liés aux types.

Nous avons aussi procédé à un encapsulage technique dans le module TR_listeen regroupant l'IP, le masque et l'interface dans un enregistrement Regle. Cela permet d'avoir accès plus facilement à l'information souhaitée.

5. Principaux algorithmes et types de données

A) Module "TR_Liste"

-> *Traiter une adresse IP qui arrive en entrée*
procedure **TraiterIP_liste**(Sdad : in T_liste; IPentree : in T_IP

-> *Est-ce qu'une adresse IP est présente dans le cache ?*
function **Cle_Presente**(Sda : T_liste ; IP : T_IP) return Boolean

-> *Obtenir la taille du cache*
function **Taille_liste**(Sda : in T_liste) return Integer

-> *Enregistrer une interface associée à une adresse IP .*
Si la clé est déjà présente dans la Sda, sa donnée est changée.
procedure **Enregistrer_liste** (Sda : in out T_liste ; IP : in T_IP ; Interfac : in T_Interface ; Masque : in T_IP)

-> *Supprimer la règle associée à l'adresse IP de destination*
Exception : Cle_Absente_Exception si Clé n'est pas utilisée dans la Sda
procedure **Supprimer_par_destination_liste** (Sda : in out T_liste ; Destination : in T_IP)

-> Savoir si une interface est présente dans une Sda.
function **La_Donnee_Inter_liste** (Sda : in T_liste ; Destination : in T_IP) return Boolean

-> Savoir si un masque est présent dans une Sda.
Exception : Cle_Absente_Exception si Clé n'est pas utilisée dans l'Sda
function **La_Donnee_Masque_liste** (Sda : in T_liste ; Destination : in T_IP) return T_Donnee

B) Module “ABR”

->Vérifier qu’un Noeud ou qu’un Arbre soit vide
function **Est_Vide** (Arbre : T_ABR) return Boolean;

-> Vérifier qu’il s’agisse bien d’une Feuille
function **Est_Feuille** (Arbre : T_ABR) return Boolean;

-> Renvoyer l’interface associée à une adresse_IP dans un Arbre
Exception : Cle_Absente_Exception si l’IP n'est pas dans l'arbre
function **La_Donnee_Inter** (Arbre : in T_ABR ; IP : in T_IP) return Interface;

-> Vérifier qu’une adresse_IP d’entrée est présente dans le cache et donc dans l’Arbre
function **Est_Presente_LA** (Arbre : in T_ABR ; IP : in T_IP) return Boolean;

-> Initialiser l’Arbre en créant un Noeud vide
procedure **Initialiser_cache_Arbre** (Arbre : out T_ABR)

-> Ajouter une Regle dans le cache et donc dans l’Arbre
procedure **Ajouter** (Arbre : in out T_ABR ; IP : in T_IP ; Interface : in T_Interface ; Masque : in T_IP)

-> Supprimer une Regle du cache en transformant une Feuille en Noeud
procedure **Supprimer** (Arbre : in out T_ABR) ;

-> Elaguer les branches inutiles de l’arbre qui le déséquilibre de façon à retrouver un arbre préfixe
procedure **Elaguer** (Arbre : in out T_ABR)

-> Renvoyer la Taille de l’Arbre ce qui nous permet de savoir si le cache est plein ou non pour appliquer la politique LRU
function **Taille** (Arbre : in T_ABR) return Integer

C) Module “CLI”

-> Lire les commandes de l'utilisateur entrées en ligne de commande lors de l'exécution du programme
procedure **Lecture_lcommande** (stats_commande : in out Integer; politique : in out integer; taille_cache : in out Integer; fichier_table : in out Unbounded_String; fichier_paquet : in out Unbounded_String; fichier_resultat : in out Unbounded_String);

-> Convertir une règle au format "IP MASQUE INTERFACE" en trois chaînes de caractères
procedure **parse_regle**(Regle: Unbounded_String ; Front_Half : in out Unbounded_String ; Middle_Half : in out Unbounded_String ; Back_Half : in out Unbounded_String);

-> Retirer des caractères nuisibles à la lecture de la chaîne
function **Remove_char**(Back_Half : in out Unbounded_String)

-> Convertir une IP au format chaîne de caractères en adresse IP au format T_Adresse_IP
function **parse_ip**(Regle: Unbounded_String)

-> Convertir une adresse IP au format T_Adresse_IP en chaîne de caractères
function **deparse_ip**(IP1: in T_Adresse_IP)

-> Savoir si Masque1 < Masque2
inf(Masque1 : T_Adresse_IP; Masque2 : T_Adresse_IP)

->Renvoyer la dernière position de 1 dans Masque en partant de la gauche du masque
Pos_dernier_1(Masque : in T_Adresse_IP) construire_masque(Taille : in Integer)

D) Type de Donnée

1) Module "**TR_Liste**"

```
-- Création du type T_Cellule, pour faire un encapsulage technique des informations techniques
T_regle_d est un ENREGISTREMENT
        destination : T_IP = T_Adresse_IP
         masque : T_IP = T_Adresse_IP
         interface : T_Interface = chaîne de caractères
Fin ENREGISTREMENT              -- ce module est implanté génériquement

-- Création du type T_Cellule, un maillon de la liste chaînée
T_Cellule est nouveau ENREGISTREMENT
        regle : T_regle,
        compteur : Entier
        Suivant : T_liste
Fin ENREGISTREMENT

-- Création du type T_liste, pointant vers un maillon de la liste chaînée
T_liste est pointeur vers T_Cellule

cache est T_liste
```

2) Module "**ABR**"

On utilise le type **T_Noeud**, **T_ABR** et **T_Regle** : cela permet de réaliser un arbre binaire constitué de Noeuds ordonnés grâce à la relation d'ordre "<" et de renvoyer facilement une règle contenant des informations comme l'adresse_IP, le masque ou l'interface.

```
->type T_ABR is access T_Noeud;
->type T_Regle is record
         IP : T_IP ;            -- on utilise la généricité en utilisant des paramètres génériques
        Masque : T_IP ;         --  qu'il nous sera possible d'instancier si l'on souhaite
        Interfac : T_Interface;   -- un type T_adresse_IP, un string, etc
end record;
->type T_Noeud is record
        Regle : T_Regle
        SAG : T_ABR         -- SAG = Sous-Arbre-Gauche
        SAD : T_ABR         -- SAD = Sous-Arbre-Droit
        bool : Boolean      -- Boolean pour reconnaître un noeud (Faux) et une Feuille (Vrai)
        VA : Entier          -- VA = Valeur d'Ancienneté, compteur pour la politique LRU
end record;
```

3) Module “**CLI**”

type T_Adresse_IP est modulo 2**32 -- Former une adresse_IP de 32 bits

Ce module contient les lignes de commande et les opérations sur les adresses IP

6. Test du programme

Nous avons effectué un **test du module TR_Liste** pour s’assurer de son bon fonctionnement puisque nous en avons ensuite besoin pour la table de routage et le cache. De plus un **test du module CLI**, nous a confirmé le bon fonctionnement des lignes de commande.
Finalement, nous avons pu **tester le routeur_LL** avec plusieurs paquets différents toujours triés correctement du masque le plus précis au plus général. En variant les lignes de commande nous avons vu que notre programme était bien fonctionnel et qu’il répondait au cahier des charges.

7. Difficultés rencontrées et solutions apportées

La première difficulté rencontrée venait de la compréhension et de la manipulation du type Adresse_IP. Pour pouvoir surpasser cela nous avons dû bien comprendre et travailler le fichier exemple_adresse_IP fournit dans le sujet qui nous a grandement aidé.

Ensuite, la deuxième difficulté venait de la gestion du cache. Nous avions bien compris qu’il fallait prendre le masque associé à une adresse IP destination l’ajouter dans le cache en prenant le masque le plus long et en l'écourtant d'un certain nombre de bit pour qu’on puisse discriminer des autres adresses IP destinations, nous avons alors pu trouver la méthode la plus optimisée pour réduire le nombre de bit du masque comme le montre le schéma d’optimisation du cache dans (PIM raffinages). Toutes les fonctions associées ont été mises au point pour pouvoir réaliser un programme routeur_LL avec optimisation du cache. Cependant cette méthode a été trouvée assez tardivement, même si nous avons terminé le programme routeur_LL nous aurions pu donner une forme très optimisée en travaillant à 3 sur le projet.

Enfin, la dernière difficulté réside dans la compréhension et la manipulation du module “ARB” qui nous a demandé un certain temps de compréhension. Mais nous avons pu surmonter ce problème en se documentant auprès des professeurs et sur internet. Nous avons choisi de nous concentrer sur des raffinages solides et complets témoignant de notre compréhension de ce module. Nous avons pu mettre au point la quasi-totalité des fonctions associées à ce module mais nous n’avons pas pu l’implémenter par faute de temps.

8 Organisation et répartition du travail

Nous avons commencé par réfléchir chacun de notre côté sur les raffinages de la table de routage et de la gestion du cache, avant de mettre nos idées en commun. Robin s’est chargé du raffinage du routeur_LL, Mathieu du routeur_LA et Vithursan de la table de routage. Par la suite, nous avons commencé l’écriture des modules “TR” et “LCA”. Robin s’est occupé de la partie main tandis que Mathieu s’est occupé de la partie ligne de commande et test du routeur simple. Vithursan s’est occupé de la gestion du cache. Ensuite Robin s’est chargé du routeur_LL avec l’implantation du module “TR_Liste” , Mathieu s’est occupé de l’implantation du module “ABR” et nous avons tous contribué à la rédaction des livrables, certains plus que d’autres.

9. Bilan technique

● Bilan technique : Au travers de ce projet nous avons pu renforcer nos compétences en programmation impérative dans des domaines comme la généricité ou l’utilisation spécifique des modules. Ce projet riche et complet nous a aussi donné des notions en réseau en réalisant un routeur_LL est fonctionnel.

● Etat d’avancement du projet : Le routeur_LL répond au cahier des charges. Le projet n’est pas loin d’être terminé avec quelques fines modifications dans les algorithmes gérant l’optimisation du cache et dans le module de l’arbre.

● Perspectives d’amélioration : L’implémentation du cache optimisée du routeur_LL et l’implémentation du module ABR.

10 Bilan personnel et individuel

10.1 Perspective de Mathieu ZEIDLER

Ce projet m'a paru au début compliqué car je n'avais aucune notion en réseau. Il fallait comprendre le principe du routeur, de la gestion des caches et du module ABR. Néanmoins, en persévérant et avec l'aide précieuse de nos professeurs, on avance petit à petit pour finalement délivrer un projet riche et complexe. J'ai remarqué l'importance d'une bonne organisation et communication dans l'équipe. De plus, dans un projet à plusieurs, l'avancée du projet dépend également des autres membres si chacun y met de la bonne volonté, on peut toujours y arriver. Malheureusement Vithursan a été inactif pendant ce projet, de ce fait nous avons eu une charge de travail beaucoup plus importante ainsi qu'un projet non complètement abouti. On apprend à gérer nos problèmes ensemble et à réfléchir à plusieurs, même s'il ne faut pas toujours compter sur les autres. J'ai passé énormément de temps sur ce projet, surtout pour comprendre le fonctionnement de l'arbre et pour réaliser tous les livrables. Je ressors cependant grandi de ce projet grâce à une bonne cohésion avec Robin.

10.2 Perspective de Robin Augereau

Le début du projet a été difficile car le sujet était difficilement compréhensible. Les notions de “et logique” et de masque, de sous-réseau sont abstraites si on ne connaît pas le monde du réseau. Le projet était assez long avec un travail mal réparti entre les membres du groupe car tout le monde n'a pas fourni la même quantité de travail. Vithursan a écrit un total de 40 lignes de code au long de ce projet dont aucune ne compile. Cependant, ce projet a été très enrichissant, avec un perfectionnement personnel de l'utilisation des routeurs et de l'utilisation de modules. Le sujet inclut énormément de notions différentes, et de contraintes techniques, ce qui le rend vraiment intéressant à comprendre et à implanter. Merci à Mathieu pour son investissement.

10.3 Perspective de Vithursan MURUGESAPILLAI

Le projet que nous avons m'a permis d'acquérir une multitude de compétences. Le fait d'être confronté à un problème qui ne faisait guère partie de mes connaissances m'a poussé à développer des méthodes de travail et de recherche. La charge de travail et d'acquisition et de compréhension des connaissances liées au projet était présente. La recherche de solution a été active durant la période du projet, et la complexité liée à l'optimisation du cache était déterminante, mais la solution a été trouvée assez tardivement. Ceci m'a appris que l'organisation du travail est primordiale dans tout projet à entreprendre afin d'aller vers la réussite. J'ai également énormément appris en termes de connaissances informatiques et j'en sors grandi et fier de ce projet que j'ai pu réaliser avec mes camarades.