

# **Projet PIM : Stockage et exploitation de tables de routage**

Auchère, Charles, Pisenti, Trichard

Département Sciences du Numérique - Première année  
2022-2023

## Table des matières

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Résumé</b>	<b>3</b>
<b>3</b>	<b>Architecture en modules</b>	<b>4</b>
<b>4</b>	<b>Principaux choix réalisés</b>	<b>6</b>
<b>5</b>	<b>Principaux algorithmes et types de données</b>	<b>9</b>
5.1	Principaux algorithmes . . . . .	9
5.2	Principaux types de données . . . . .	12
<b>6</b>	<b>Démarche adoptée pour tester le programme</b>	<b>14</b>
<b>7</b>	<b>Raffinages</b>	<b>14</b>
<b>8</b>	<b>Difficultés rencontrées et les solutions adoptées</b>	<b>15</b>
<b>9</b>	<b>Organisation de l'équipe</b>	<b>16</b>
<b>10</b>	<b>Bilan technique</b>	<b>17</b>
<b>11</b>	<b>Bilan personnel</b>	<b>18</b>
11.1	Nathan Auchère . . . . .	18
11.2	Ines Charles . . . . .	18
11.3	Téo Piseni . . . . .	19
11.4	Matthieu Trichard . . . . .	19

## 1 Introduction

Un routeur est un élément d'un réseau qui permet de transmettre les paquets reçus sur une interface d'entrée vers la bonne interface de sortie selon les informations qui sont stockées dans sa table de routage. Dans cette optique, notre premier objectif a été d'implanter un routeur dit simple, c'est-à-dire un routeur qui conserve toutes les routes dans une structure de donnée, nous permettant ainsi de retrouver l'interface de sortie à utiliser selon la destination. Par la suite, nous avons amélioré l'efficacité de notre routeur en utilisant un cache conservant des informations nécessaires dans la trouvaille de la bonne sortie. Pour cela, nous avons implanté des caches avec différentes politiques d'utilisations pour que le cache ne soit pas trop grand et pour un large choix d'utilisation.

Dans un premier temps nous décrirons l'architecture de l'application. Ensuite nous présenterons les principaux choix qui ont été réalisés ainsi que les principaux algorithmes et types de données. Nous décrirons par la suite notre démarche de test des programmes et nos principales difficultés et les solutions que nous avons trouvées pour les résoudre. Nous indiquerons également notre organisation d'équipe. Ce rapport sera conclu par un bilan technique du projet et un bilan personnel proposé par chacun des membres du groupe.

## 2 Résumé

Tout au long de ce projet, nous avons eu pour objectif de réaliser un routeur simple puis un routeur avec un cache. Le développement a été réalisé en respectant la méthode des raffinages pour une meilleure vision de la division du travail par la suite. Des tests unitaires ont été mis en place pour garantir le fonctionnement global. Enfin, à l'heure actuelle, nous avons un routeur qui fonctionne avec un cache pour les différentes politiques souhaitées.

### 3 Architecture en modules

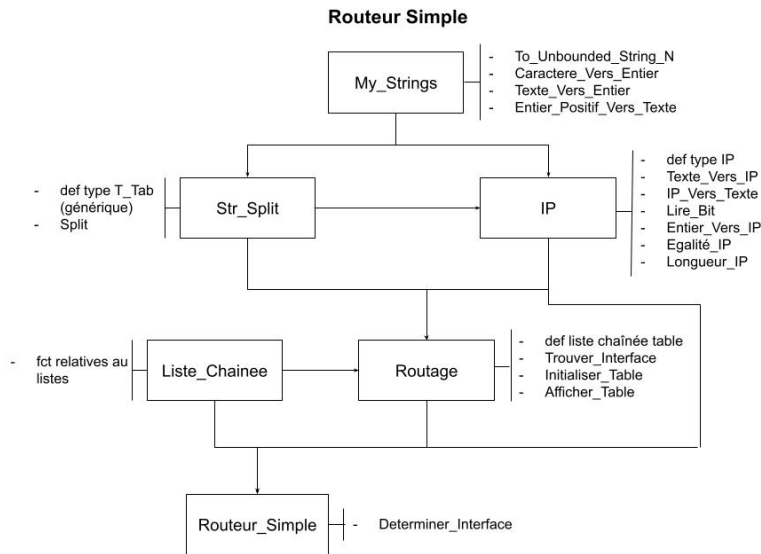


FIGURE 1 – Architecture en module du routeur simple

Pour ce routeur, nous avons créés plusieurs modules qui permettent d’effectuer des actions simples sur les types *strings*, *IP* et *Liste\_Chainee*. Ainsi, nous avons juste à les appeler dans le module *Routage* qui est le module qui va trouver l’interface et afficher la table. Enfin, nous avons le module *Routeur\_Simple* qui permet d’analyser la ligne de commande à l’aide des modules *Liste\_Chainee* et *Routage*.

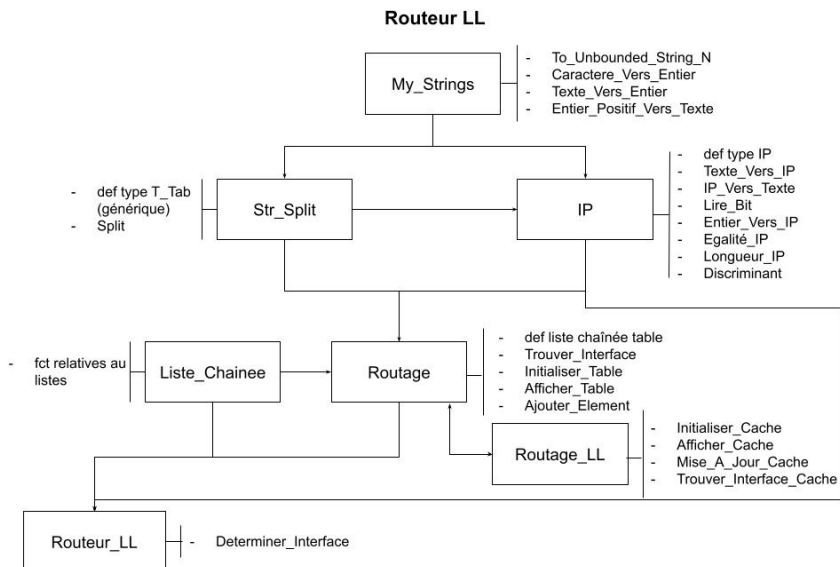


FIGURE 2 – Architecture en module du routeur avec un cache

Pour ce routeur, nous avons créés plusieurs modules qui permettent d’effectuer des

actions simples sur les types *strings*, *IP* et *Liste\_Chainee*. Ainsi, nous avons juste à les appeler dans le module *Routage* qui est le module qui va trouver l'interface, afficher la table mais aussi gérer le cache à l'aide du module *Routage\_LL*. Enfin, nous avons le module *Routeur\_LL* qui permet d'analyser la ligne de commande à l'aide des modules *Liste\_Chainee* et *Routage\_LL*. Pour ce routeur, nous avons créés plusieurs modules qui

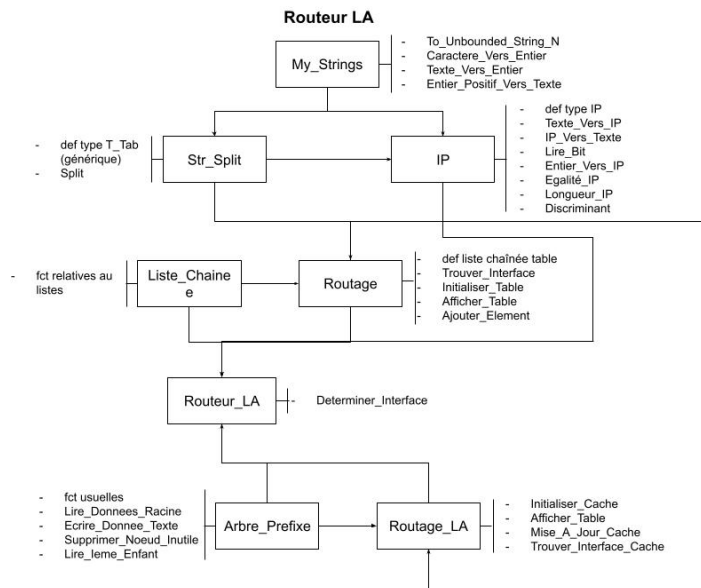


FIGURE 3 – Architecture en module du routeur LA

permettent d'effectuer des actions simples sur les types *strings*, *IP* et *Liste\_Chainee*. Ainsi, nous avons juste à les appeler dans le module *Routage* qui est le module qui va trouver l'interface et afficher la table. De plus, le module *Routage\_LA* avec l'aide du module *Arbre\_Prefixe* qui permet de s'occuper des arbres du cache, permet de gérer le cache du routeur. Enfin, nous avons le module *Routeur\_LL* qui permet d'analyser la ligne de commande à l'aide des modules *Liste\_Chainee* et *Routage\_LA*.

## 4 Principaux choix réalisés

**Système indice dans l'arbre** Afin de savoir quel élément est : le plus ancien, le moins récemment utilisé, le moins fréquemment utilisé, on a décidé de rajouter un paramètre Id de type *Natural* tel que, quel que soit la politique utilisée, l'élément à supprimer en cas de dépassement de la capacité du cache soit celui avec la valeur de Id minimale.

- FIFO : Id contient le numéro de la i-eme insertion à laquelle la route a été ajoutée au cache
- LRU : Id contient le numéro de la i-eme utilisation réussie du cache pour laquelle la route a été utilisé
- LFU : Id contient la fréquence d'utilisation de la route

### **Garder une fréquence pour tout type de cache dans le cas LL**

Lors de la création du type *T\_Cache*, 2 solutions se sont offertes à nous. La première consistait à réutiliser le type *T\_Table* pour les politiques FIFO et LFU et utiliser un type *T\_Cache* distinct avec la fréquence en plus dans *T\_Element*, ou bien d'ajouter la fréquence pour toute politique. Le premier choix permet de sauver de la RAM lors de l'exécution du programme, puisque hors politique LFU chaque cellule contient un entier de moins. Néanmoins ce gain est minime et nous avons estimé que pour un routeur, de nos jours, l'écart de RAM engendré par le second choix est négligeable sur la facilité d'implémentation, de lisibilité et de compréhensibilité du code.

### **Limite théorique des indices de l'arbre assez élevée mais peut être améliorée**

Dans la même lignée que le choix précédent, nous avons choisi d'affecter à chaque route enregistrée dans l'arbre un indice qui lui est propre. Cela implique que le routeur finira par atteindre la limite théorique des indices, leur type étant *Natural*, cette limite est de  $2^{16}$ . Puisque cela prendra plus d'une année à atteindre par un routeur recevant 10 requêtes par secondes une fois le cache plein, nous avons décidé que ce problème était négligeable. Une solution possible pour régler ce problème est de "remettre à 0" les indices à partir d'une certaine limite. C'est à dire qu'au delà d'un seuil fixé (indice  $> 2^{15}$  par exemple), un parcours de l'arbre serait lancé de telle sorte que le plus petit indice soit remis à 1 et ainsi de suite en conservant l'ordre déjà établi des indices.

### **Choix du cache : liste chaînée ou tableau**

Nous avons envisagé de multiples implémentations possible pour le cache LL.

- une liste doublement chaînée : algorithmes plus performants pour le tri du cache (pas besoin de reparcourir depuis le début à chaque fois). Algorithmes beaucoup

plus difficile à écrire (on doit mettre à jour jusqu'à 4 pointeurs dans le bon ordre lors de l'utilisation)

- une liste simplement chaînée avec stockage de l'indice du dernier élément (un FIFO) : algorithme en temps constant pour ajouter/supprimer des éléments dans le cas de la politique FIFO. Pas d'avantage notable pour les autres politiques (LFU,LRU)
- un tableau : cela est possible puisque le cache a une capacité maximale connue. algorithme plus simples (pas besoin de gérer dynamiquement la mémoire) mais moins performants (même si on conserve une classe de complexité linéaire)
- une liste simplement chaînée : choix retenu, car imposé par le sujet, et permet un bon équilibre entre complexité temporelle, spatiale et pour écrire le code

### **Choix private ou limited**

- Pour les structures chaînées, on a choisi de les mettre en type "limited private" afin de s'assurer de garder une cohérence des listes, du cache et d'éviter des erreurs de la part de l'utilisateur. Cependant, cela nous impose d'écrire beaucoup plus de procédures et de remplacer beaucoup de fonctions par des procédures (puisque'on ne peut plus faire d'affectations), ce qui est plus lourd à manipuler. Le choix limited nous a également imposé d'écrire beaucoup de procédure qui prennent en paramètre des procédures génériques afin de pouvoir réaliser certaines actions n'ayant pas été pensées par le développeur originel (voir module Arbres\_Prefixes).
- Pour le type T\_IP, on a choisi de mettre le type en "private" et non "limited private" car on considère qu'il est pertinent de pouvoir faire des affectations et des comparaisons d'adresses IP dans le code des utilisateurs. (le type T\_IP étant relativement simple, et ne contenant pas d'allocation dynamique par exemple qui pourraient permettre à l'utilisateur de faire des choses incohérentes contrairement à précédemment. Par exemple)

### **Choix de l'arbre préfixe avec plusieurs fils et versatile**

Nous avons décidé d'implémenter un module *Arbre\_Prefixe* générique pour la gestion du cache LA. Ce module est capable de gérer tout type d'arbres préfixes avec une Arité/Nombre de Fils arbitraire. Il peut être utilisé pour modéliser/reconnaître des langages avec des alphabets génériques (un mot étant considéré terminal s'il mène jusqu'à une feuille de l'arbre). Pour parvenir à ses fins, ce module prend comme paramètre de généricité une fonction *Lire\_Préfixe* donnée par l'utilisateur qui permet d'associer à chaque symbole de l'alphabet considéré, un unique entier de 1..Nombre\_Prefixes.

Dans le cadre de ce projet, nous devons considérer le langage des adresses IP constitué de l'alphabet "0,1". Nous allons donc l'instantier avec *Nombre\_Prefixes* = 2 (arbre binaire) et "*Lire\_Préfixe*(IP,Position) = *Lire\_Bit*(IP,Position) + 1"

Afin de convenir à un nombre maximal de cas d'usages, la majorité des procédures du module prennent des procédure génériques "Post\_Traitement" en paramètre de généricité

(et doivent donc être instanciées) qui permet à chaque fois d'appliquer un `Post_Traitement` aux différents nœuds visités par la procédure en question (après que celle-ci se soit terminée, et dans l'ordre de bas en haut). Cela nous sera utile pour maintenir des relations entre les nœuds et leurs fils après chaque modification.

Ce module, bien que suffisant pour notre projet, reste cependant non exhaustif des fonctions sur les `Arbres_Prefixes`. Et peut être étoffé de nouvelles procédures lors d'utilisation dans de futurs projets.

### **Choix de plusieurs politiques pour le cache LA**

Nous avons également pris le temps dans ce projet d'ajouter au module `Routage_LA` la gestion des trois types de cache (LFU, LRU, FIFO) bien que seul la politique LRU soit exigée par le sujet. Nous avons également écrit une fonction `Trouver_Interface_Table()` qui est inutilisée dans ce projet, mais qui aurait pu servir pour trouver l'interface dans notre arbre binaire, si celui-ci était utilisé directement comme table de routage plutôt que comme un cache. La différence majeure étant que le cache est plus strict sur les masques (en raison de la gestion de la cohérence du cache), tandis que pour une table de routage, on est obligé de parcourir la totalité de l'arbre afin de s'assurer de choisir le masque le plus long en cas de plusieurs interfaces possibles. (On constate d'ailleurs que dans ce cas les arbres binaires n'apportent pas d'avantage pour la complexité temporelle. C'est pour cela qu'ils sont utilisés uniquement pour le cache dans ce projet).

### **Choix des adresses IP cherchées**

Nous avons décidé de mettre dans le cache directement les adresses IP que l'on a routé, plutôt que de masquer celles-ci pour qu'elles se finissent pas des "...0.0" comme montré dans le sujet.

Nous avons effectivement jugé cela non utile puisque les bits restants ne sont de toute manière jamais regardés par l'ensemble des fonctions du programme.



## 5 Principaux algorithmes et types de données

### 5.1 Principaux algorithmes

#### **Routeur\_Simple.adb**

*Determiner\_Interface* : traite les instructions de gestion du routeur de la ligne de commande.

#### **Routage.adb**

*function Trouver\_Interface*(*Table\_Routage* : in *T\_Table*; *IP* : in *T\_IP*) : trouve l'interface correspondant à une IP dans la table de routage

*procedure Initialiser\_Table\_Vide*(*Table\_Routage* : *T\_Table*; *Fichier* : *File\_Type*) : initialise la table de routage avec le fichier dédié

*procedure Afficher\_Table*(*Table\_Routage* : *T\_Table*) : affiche tous les éléments de la table de routage

*function Est\_Vide*(*Table\_Routage* : *T\_Table*) : renvoie si la table est vide ou non

Pour le routeur Cache, on ajoute :

*function Determiner\_Masque\_Cache* (*Table* : *T\_Table*; *IP\_A\_Router* : *T\_IP*) : détermine le masque à mettre dans le cache

#### **Liste\_Chaine.adb**

Fonctions relatives aux listes chaînées

#### **IP.adb**

*function Texte\_Vers\_IP*(*Chaine* : in *Unbounded\_String*) : convertit une chaîne du type 192.168.0.1 en ip

*function IP\_Vers\_Texte*(*IP* : *T\_IP*) : convertit une ip en une chaîne du type 192.168.0.1

*function Lire\_Bit*(*IP* : *T\_IP*; *i* : *Integer*) : renvoie la valeur du i-ème bit (en partant de la gauche)

*function Entier\_Vers\_IP*(*n* : *Integer*) : convertit un entier en ip (on ne peut pas représenter toutes les ip par des entiers, car ils sont trop petits)

*function Egalite\_IP*(*IP1* : *T\_IP*; *IP2* : *T\_IP*; *Masque* : *T\_IP*) : teste l'égalité entre 2 ip pour un masque donné

*function Longueur\_IP*(*IP* : *T\_IP*) : renvoie la longueur max de 1 dans ip en partant de la gauche

*function Discriminant*(*IP1* : *T\_IP*; *IP2* : *T\_IP*) : renvoie le masque de longueur minimale qui discrimine IP1 et IP2

#### **Str\_Split.adb**

*procedure Split*(*Tableau* : *T\_TAB*; *Chaine* : *Unbounded\_String*; *Caractere* : *Charac-*

*ter*) : permet de couper une chaîne sur le caractère *c* et remplit un tableau de *NbrArgs* éléments. Si on a plusieurs fois *c* d'affilé dans la chaîne, on ne coupe qu'une seule fois

### **My\_Strings.adb**

*function To\_Unbounded\_String\_N(Chaîne : String ; n : Integer)* : créer un *Unbounded\_String* avec les *n* premiers caractères de chaîne

*function Caractere\_Vers\_Entier(c : Character)* : convertit un caractère en chiffre

*function Texte\_Vers\_Entier(Texte : String)* : convertit une string en un entier, les espaces sont ignorés et respecte le signe

*function Entier\_Positif\_Vers\_Texte(n : Integer)* : convertit un entier positif en texte (sans rajouter d'espace avant ou après le nombre)

### **Routeur\_LL.adb**

*Determiner\_Interface* : traite les instructions de gestion du routeur de la ligne de commande.

### **Routage\_LL.adb**

*procedure Initialiser\_Table\_Vide (Table\_Routage : T\_Table)* : initialise avec le cache vide

*procedure Vider\_Table (Table\_Routage : T\_Table)* : vide le cache

*procedure Trouver\_Interface\_Cache (Interface\_Nom : Unbounded\_String ; Table\_Routage : T\_Table ; IP : T\_IP ; Politique\_Cache : Unbounded\_String)* : trouve l'interface correspondant à une IP dans la table de routage

*procedure Afficher\_Cache(Cache : T\_Table ; Politique\_Cache : Unbounded\_String)* : affiche tous les éléments du cache

*function Est\_Vide (Table\_Routage : T\_Table)* : renvoie si le cache est vide ou non

*procedure Mise\_A\_Jour\_Cache(Cache : T\_Table ; IP\_A\_Router : T\_IP ; Capacite\_Cache : Integer ; Politique\_Cache : Unbounded\_String ; Taille\_Cache\_Actuelle : Integer ; Interface\_Nom : Unbounded\_String ; Table : Routage.T\_Table)* : met à jour le cache en accord avec la politique et la cohérence du cache

### **Routeur\_LA.adb**

*Determiner\_Interface* : traite les instructions de gestion du routeur de la ligne de commande.

### **Routage\_LA.adb**

*procedure Initialiser\_Table\_Vide (Table\_Routage : T\_Table)* : initialise avec le cache vide

*procedure Vider\_Table (Table\_Routage : T\_Table)* : vide le cache

*procedure Trouver\_Interface\_Table (Interface\_Nom : Unbounded\_String ; Table\_Routage : T\_Table ; IP : T\_IP)* : trouve l'interface correspondant à une IP dans la table de routage

*procedure Trouver\_Interface\_Cache (Interface\_Nom : Unbounded\_String ; Table\_Routage : T\_Table ; IP : T\_IP ; Politique\_Cache : Unbounded\_String) : trouve l'interface correspondant à une IP dans la table de routage. Cette fonction ne fonctionne que dans le cas où la Table\_Routage est en réalité un cache LA*

*procedure Afficher\_Table (Cache : T\_Table ; Politique\_Cache : Unbounded\_String) : affiche tous les éléments de la table de routage*  
*function Est\_Vide (Table\_Routage : T\_Table) : renvoie si la table est vide ou non*

*procedure Mise\_A\_Jour\_Cache (Cache : T\_Table ; IP\_A\_Router : T\_IP ; Capacite\_Cache : Integer ; Politique\_Cache : Unbounded\_String ; Taille\_Cache\_Actuelle : Integer ; Interface\_Nom : Unbounded\_String ; Table : Routage.T\_Table) : met à jour le cache en accord avec la politique et la cohérence du cache*

### **Arbre\_Prefixe.adb**

*procedure Initialiser (Arbre : T\_Trie) : initialise l'arbre*

*procedure Vider (Arbre : T\_Trie) : vide l'arbre*

*function Est\_Vide (Arbre : T\_Trie) : renvoie si l'arbre est vide ou non*

*function Est\_Feuille (Arbre : T\_Trie) : renvoie si l'élément est une feuille*

*procedure Ajouter (Arbre : T\_Trie ; Cle : T\_Cle ; Element : T\_Element) : Ajouter un élément dans l'arbre que l'on peut retrouver en suivant successivement le chemin (regarder récursivement dans les bons fils) associé aux lettres de la clé donnée par la fonction Lire\_Préfixe*

*function Lire\_Donnee\_Racine (Arbre : T\_Trie) : lit la donnée contenue dans la racine de l'arbre passé en paramètre*

*function Lire\_Cle\_Racine (Arbre : T\_Trie) : lit la clé associée à la donnée contenue dans la racine de l'arbre passé en paramètre*

*function Ecrire\_Donnee\_Tete (Arbre : T\_Trie ; Donnee : T\_Element) : modifie la donnée contenue dans la racine de l'arbre passé en paramètre*

*function Lire\_Ieme\_Enfant (Arbre : T\_Trie ; i : Natural) : Obtient une copie du i-eme enfant de l'arbre courant*

*procedure générique Supprimer\_Selection (Arbre : T\_Trie) avec procedure Post\_Traitement (Arbre : T\_Trie) et function Selection (Arbre : T\_Trie) : supprime toutes les feuilles de l'arbre F telles que Selection(F) = True. La fonction générique "Selection" renvoie True si le nœud courant est un parent d'au moins un nœud à supprimer. Elle renvoie également True si le nœud courant est une feuille à supprimer de l'arbre. False dans les autres cas. La fonction Post\_Traitement permet d'appliquer un post traitement sur tous les nœuds visités après la suppression des éléments à supprimer. Et dans l'ordre inverse dans lequel ils ont été visité.*

*procedure générique Parcours\_Profondeur\_Post (Arbre : T\_Trie) : applique un Traitement à tous les nœuds de l'arbre, en suivant l'ordre d'un parcours en profondeur*

*procedure Chercher\_Et\_Verifier\_Post (Element\_Trouve : T\_Element ; Arbre : T\_Trie ; Cle : T\_Cle) : cherche une clé dans l'arbre en suivant la méthode habituelle des arbres*

préfixes (À chaque profondeur. on choisit le fils correspondant au caractère actuel de la clé renvoyé par la fonction Lire\_Prefixe). Quand on arrive sur la feuille finale, on regarde si le mot trouvé correspond aux attentes

## 5.2 Principaux types de données

Pour le routeur simple, nous avons créé les types de données suivants :

*type privé  $T\_IP$  : le type de l'adresse IP définit comme  $mod2 * 32$*

*type Generique  $T\_Table$  : la table de routage de notre routeur*

C'est un pointeur sur  $T\_Cellule$

*type  $T\_Cellule$  : c'est un enregistrement contenant les informations suivantes :*

- *Adresse* : l'adresse IP souhaitée de type  $T\_IP$  ;
- *Masque* : le masque obtenu à partir de l'IP de type  $T\_IP$
- *Interface\_Nom* : nom donnée à l'interface résultant de cette adresse et de ce masque de type *Unbounded\_String*

Pour le routeur avec un cache (LL) :

*type privé  $T\_IP$  : le type de l'adresse IP définit comme  $mod2 * 32$*

*type Generique  $T\_Cache$  : le cache de notre routeur*

C'est un pointeur sur  $T\_Cellule\_Cache$

*type  $T\_Cellule\_Cache$  : c'est un enregistrement contenant les informations suivantes :*

- *Adresse* : l'adresse IP souhaitée de type  $T\_IP$  ;
- *Masque* : le masque obtenu à partir de l'IP de type  $T\_IP$
- *Interface\_Nom* : nom donnée à l'interface résultant de cette adresse et de ce masque de type *Unbounded\_String*
- *Frequence* : fréquence associée à la cellule type *Integer*

Pour le routeur avec un cache (LA) :

*type privé  $T\_IP$  : le type de l'adresse IP définit comme  $mod2 * 32$*

*type Generique T\_Trie* : le cache de notre routeur  
C'est un pointeur sur *T\_Noeud*

*type T\_Noeud* : c'est un enregistrement contenant les informations suivantes :

— *Cle* : la cle du noeud *T\_Cle*

— *Element* : l'élément du noeud type *T\_Cellule*

— *Enfants* : représente les fils du noeud, avec pour type un tableau allant de 1 à *Nombre\_Enfant\_Max* de *T\_Trie*

*type T\_Cellule* : c'est un enregistrement contenant les informations suivantes :

— *Masque* : le masque obtenu à partir de l'IP de type *T\_IP*

— *Interface\_Nom* : nom donnée à l'interface résultant de cette adresse et de ce masque de type *Unbounded\_String*

— *Id* : représente le fils le moins récemment utilisé type *Natural*

## 6 Démarche adoptée pour tester le programme

Dans un premier temps, nous avons créé des fichiers de test pour tous les modules que nous étions en train de coder. Ainsi, nous pouvions vérifier au fur et à mesure que nos programmes fonctionnaient correctement. Nous retrouvons ces fichiers de test sous le nom *test\_* puis le nom du module testé. Dans chacun de ces fichiers, il y a une partie dédiée à chacune des fonctions implantées dans le module.

Dans un second temps, des tests unitaires ont été réalisés notamment pour les routages, que ce soit le routeur simple ou le routeur avec un cache. Pour cela, nous avons écrit de nombreuses lignes de commandes et nous avons vérifié que le programme réagissait de la façon souhaitée.

## 7 Raffinages

Vous trouverez donc le dossier un fichier nommé *Raffinages.pdf*. Ce fichier contient nos raffinages ainsi que notre grille d'auto-évaluation.

## 8 Difficultés rencontrées et les solutions adoptées

### **Familiarisation avec le git**

Un point difficile à aborder lors de ce projet fût l'apprentissage de l'utilisation de git à plusieurs. Bien que la prise en main ait été finalement rapide, réussir à comprendre et gérer les conflits au départ n'a pas été évident. Ainsi nous avons appris à réfléchir aux impact des commandes, à la clarté des commit et à entretenir une sauvegarde externe.

### **Gestion d'un grand nombre de module au sein du même projet**

Ce projet se constitue en un nombre important de modules liés les uns aux autres. La création de ces modules fût une chose, mais les relier proprement et gérer les types privé et limité privé en fût une autre. En effet, cela nous a demandé des réflexions préalables à l'écriture de ces modules puisque les types définis dans les uns devaient ou non pouvoir être accessibles dans les autres.

De même, l'instanciation des différents modules et fonctions n'a pas toujours été des plus évident.

### **Trouver rapidement l'élément à supprimer**

Nous avons été soumis à un problème lors de l'implémentation des différentes politiques du cache LA. Il fallait pouvoir trouver rapidement l'élément d'indice minimal quand le cache est plein afin de le supprimer. Afin d'éviter de parcourir tout l'arbre (ce qui ferait perdre l'avantage de l'utilisation de structures d'arbres avec une complexité très élevée), nous avons décidé de conserver dans chaque nœud intermédiaire de l'arbre, le numéro "Id" le plus petit de tous ces nœuds enfants.

Ainsi on connaîtra directement la valeur de l'Id minimal en regardant celui de la racine de l'arbre, et on pourra chercher uniquement dans les branches qui conservent cet Id minimal plutôt que de tout parcourir.

## 9 Organisation de l'équipe

Quoi ?		Qui ?			
Modules	Sous-programmes	Spécification	Codage	Test	Relecture
Routeur Simple	Determiner l'interface	I	NA	M	M
Routage	Trouver_Interface	M	NA	M	I
	Initialiser_Table	NA	M	I	M
	Ajouter_Element	M	I	TP	NA
	Afficher_Table	I	M	I	NA
IP	Texte vers IP	TP	TP	NA	I
	IP vers Texte	TP	TP	I	NA
	Lire un bit	TP	TP	NA	I
	Entier_Vers_IP	TP	NA	I	TP
	Longueur_IP	I	M	NA	I
	Egalité entre 2 Ip avec masque	TP	NA	I	TP
	Discriminant	I	NA	TP	M
Str_Plit	Split	I	TP	NA	M
My_Strings	To Unbounded_String_N	M	I	M	NA
	Caractere_Vers_Entier	M	I	NA	TP
	Texte_Vers_Entier	NA	I	TP	M
	Entier_Positif_Vers_Texte	NA	M	TP	TP
Liste Chainees	Fonctions des listes	NA	I	M	TP
Routeur_LL	Determiner l'interface	I	NA	M	TP
Routage LL	Initialiser_Cache_Vide	I	NA	M	TP
	Trouver_Interface_Cache	NA	I	M	TP
	Afficher_Cache	NA	I	TP	M
	Mise_A_Jour_Cache	I	NA	TP	M
Routeur LA	Determiner l'interface	M	TP	I	NA
Arbre_Prefixe	Fonctions usuelles	M	TP	I	NA
	Lire_Donnee_Racine	M	TP	I	NA
	Ecrire_Donnee_Tete	TP	M	I	NA
	Supprimer_Noeud_Inutile (private)	TP	M	NA	I
	Lire_1eme_Enfant	TP	M	NA	I
Routage LA	Initialiser_Cache	M	TP	NA	I
	Trouver_Interface_Cache	M	TP	NA	I
	Afficher_Table	TP	M	NA	I
	Mise_A_Jour_Cache	TP	M	NA	I



## 10 Bilan technique

Au moment de ce rendu, nous avons implémenté le routeur simple mais aussi le routeur avec une cache. Les deux types de cache ont été implémentés, le routeur LL qui a pour cache une liste chaînée et le routeur LA qui a pour cache un arbre préfixe.

Comme il l'était subtilement pointé dans le sujet, nous avons opté pour une implémentation de choix du masque dans le cache le plus précis. Autrement que d'aller au bout de l'idée de la cohérence de cache, ce choix permet un routage "plus efficace" par le fait que le cache est entièrement discriminant. C'est à dire qu'il contient le plus de route possibles différentes. C'est ainsi qu'il est efficace, permettant d'éviter le plus possible le parcours peu efficace de la table. De plus, à l'heure actuelle le cache LL est trié pour avoir affichage plus clair à la suite de son changement.

De plus, comme expliqué dans la rubrique des choix effectués, nous avons une limite théorique des indices de l'arbre qui est assez élevée mais qui pourrait être améliorée. C'est une partie qui pourrait être implantée comme évolution du projet.

## 11 Bilan personnel

### 11.1 Nathan Auchère

Temps total :  $\sim 30h$

— conception :  $\sim 10h$

— implementation :  $\sim 8h$

— mise au point :  $\sim 10h$

— rapport :  $\sim 2h$

Ce projet à pour moi été intéressant sur plusieurs points. Tout d’abord, le travail d’équipe nécessaire pour le mener à bien a été un redécouverte avec l’utilisation de git et les discussions techniques. En effet les choix que le groupe a été mené à prendre on dû être débattu et considérés. Cette expérience m’a permis d’acquérir en confiance et esprit critique, écouter les arguments des uns et des autres pour se faire son avis. Ensuite, de manière plus anecdotique, j’ai apprécié comprendre et recoder le principe de fonctionnement d’un routeur, une machine présente au quotidien. Enfin, compléter petit à petit le projet, assembler chaque module avec chaque module et voir le résultat compiler et fonctionner a été un vrai plaisir.

### 11.2 Ines Charles

Temps total :  $\sim 30h$

— conception :  $\sim 6h$

— implementation :  $\sim 7h$

— mise au point :  $\sim 10h$

— rapport et autres livrables :  $\sim 7h$

J’ai trouvé ce projet très intéressant. Tout d’abord, le travail d’équipe m’a permis de mieux appréhender certaines notions et d’avoir des dialogues avec d’autres pour réaliser les choix les plus pertinents. De plus, la découverte de l’utilisation du git et ses spécificités m’a beaucoup apporté pour le futur. Ce projet m’a appris à être à l’écoute de mes camarade pour essayer d’être le plus efficace dans le travail mais surtout pouvoir écouter les points de vus de chacun et se mettre d’accord sur la bonne démarche. Ensuite, j’ai trouvé le sujet plutôt intéressant car il était en lien direct avec notre quotidien. Enfin, voir le bon fonctionnement du projet est un vrai plaisir.

### 11.3 Téo Pisenti

Temps total :  $\sim 40h$

— conception :  $\sim 15h$

— implementation :  $\sim 15h$

— tests :  $\sim 4h$

— rapport :  $\sim 1h$

— relectures et corrections d'erreurs :  $\sim 5h$

Ayant déjà mené des projets, celui-ci demeure toutefois le plus long d'entre eux (environ 3000 lignes, tout compris). La nouveauté fut également pour moi le travail en équipe et la découverte de git dont je n'avais jamais eu l'utilité avant. J'ai beaucoup œuvré à l'élaboration de l'architecture du projet, au choix des structures utilisées, aux spécification et écriture des modules primaires (indépendant de tous les autres modules) afin d'offrir à mes camarades des outils pratiques pour la suite du projet. La difficulté fut notamment de choisir les types d'accès ("limited", "private") les plus stricts prévenir les erreurs de manipulation des données. Ce qui nécessite d'écrire des procédures suffisamment versatiles pour ne pas engendrer de blocage dans la suite du projet sans influencer sur l'efficacité des algorithmes finaux. Le tout en ayant des modules aux rôles clairement identifié (exemple : pas de fonctions de routage dans le module Arbres\_Prefixes) Je pense être parvenu aujourd'hui à un compromis qui me satisfait, et c'est aussi pourquoi je suis fier de notre travail à tous les quatre. J'espère que vous parviendrez à apprécier le temps et les efforts que nous avons consacré dans ce projet.

### 11.4 Matthieu Trichard

Temps total :  $\sim 25h$

— conception :  $\sim 5h$

— implementation :  $\sim 4h$

— tests :  $\sim 15h$

— rapport :  $\sim 1h$

Pour ma part, ce projet a été très intéressant car il traite d'un système qui est omniprésent dans notre monde moderne. Il m'a permis de mieux le comprendre et d'appréhender son fonctionnement en profondeur. De plus, j'ai pu comprendre de nouvelle méthode de programmation grâce à mes partenaires. De plus, ce projet en groupe m'a permis d'apprendre à utiliser git pour les projets de code collaboratifs. Il m'a aussi permis d'apprendre à partager des idées à un groupe mais aussi à débattre des différentes propositions.