Rapport Projet PIM

Jérémy Angulo - Alexandre Trotel - FRESCO Alan - HUANG Julien

Département Sciences du Numérique - Première année
2022-2023

Table des matières

I – Introduction

I.1 Résumé

Chaque appareil connecté à Internet ou au réseau local possède une adresse IP. Il s'agit d'une adresse sur laquelle arrive les informations que vous demandez sur Internet.

Une table de routage dresse la liste des adresses IP des réseaux connus du système, notamment le réseau local par défaut. Elle répertorie également la liste des adresses IP d'un système de passerelle pour chaque réseau connu.

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.

Le prototype proposé a été programmé en langage ADA, suite à un rigoureux travail de raffinage des actions complexes composant ce projet.

I.2 Structure du document

Le chapitre II décrira le déroulé du projet, de l'analyse à sa réalisation. Ensuite, le chapitre IV décrira l'architecture du projet, c'est à dire les différents modules qui compose notre projet. Enfin, le chapitre V décrira les méthodes appliquées et les algorithmes utilisés dans la réalisation du projet.

II – Déroulé du projet

II.1 Analyse du sujet

L'analyse du sujet était difficile de prime à bord, nous n'avions rien compris pendant la première séance de projet. Cependant, l'arrivée de Jérémy a permis d'éclaircir de nombreux points ! L'utilité et le fonctionnement du routeur a été compris au fur et à mesure de l'avancement du projet notamment au cours des raffinages.

L'analyse du sujet s'est principalement basée sur des exemples. Il est plus simple de voir le chemin que prend une adresse qu'un paragraphe de texte assomant.

II.2 Organisation de l'équipe

Alexandre Je me suis occupé du routeur LA dans son ensemble. J'ai commencé par modéliser celui-ci et définir des types utilisateurs appropriés en conséquence. Ensuite, j'ai sélectionné et coder les procédures/fonctions les plus pertinentes pour le cache sous forme d'un arbre. D'un point de vue plus global, j'ai fais la partie des raffinages concernant le routeur LA. De même, je me suis occupé de cette partie dans le rapport.

Julien J'ai traité la partie sur la table de routage, et aidé à la conception des tests du module table de routage, cache_LCA, cache_LA. Rédigé la partie table de routage dans le rapport (raffinage et module) et quelques autres partie.

Jérémy J'ai établi les fondations du projet : le module tools et les trois programmes générant les trois exécutables. J'ai aussi participé à la réalisation du module table_routage avec l'aide Julien. Ayant 2 années de programmation derrière moi, je me suis rendu disponible aussi souvent que possible pour aider mes collègues dans la réalisation de leur tache. Ainsi, j'ai pu partager avec mes camarades ma façon de voir l'algorithme s'occupant de donner l'adresse, le masque et l'interface découlant du paquet envoyé, au cache.

II.3 Difficultés rencontrées

On a eu des difficultés à comprendre le cache. Heureusement les exemples du sujet nous ont permis de comprendre comment éviter les erreurs avec le cache. La compréhension complète de la programmation du projet a pris du temps. Au moment de nos raffinages, nous ne comprenions que l'objectif global, sans les détails techniques.

La réalisation de ce projet s'est réalisée en deux parties : tout d'abord le prototypage d'un routeur simple, fonctionnant sans cache, puis la réalisation de celui-ci avec cache.

Tout le long du projet, nous avons complété notre module de test, afin de s'assurer du bon fonctionnement des modules constituant le projet.

II.4 Principaux choix autour du projet

Nous avons fait le choix de dissocier la structure du cache et les paramètres de ce dernier dans le routeur LA. En effet, le type 'T_Param_Cache' a pour but de contenir les paramètres du cache comme sa capacité, sa taille et ses statistiques. D'autre part, le type 'T_Arbre' modélise la structure du cache et les informations qu'il contient. Ainsi, il faut s'assurer de mettre à jour les paramètres du cache au bon moment dans l'utilisation des procédures qui manipulent la structure du cache.

II.4.1 Test du bon fonctionnement du programme

Pour tester le programme, nous avons adoptée la démarche suivante :

Pour chaque module programmé, nous avons réalisé son module "test", regroupant toutes les procédures de test des fonctions et procédures principales du module visé. Grâce aux 'pragma Assert', nous pouvions voir facilement les comportements imprévus des modules, et corriger en conséquence. Une autre méthode de test pour le programme principal a été appliquée : nous avons créé diverses tables de routage et paquets à router, puis nous avons testé le programme en vérifiant bien la robustesse du programme et la véracité des sorties dans resultats.txt

III – Raffinage

III.1 Table de routage

R0 : Concevoir et initialiser un routeur.

R1 : Comment “Concevoir et initialiser un routeur” ?
- Concevoir un routeur
- Initialiser un routeur à partir d’un fichier texte

R2 : Comment “Concevoir un routeur” ?

```
TYPE T_Adresse_IP EST ENREGISTREMENT
    Valeur :  Entier mod 2 ** 32
    Occurence : Entier
FIN ENREGISTREMENT

TYPE T_Routeur_Cellule

TYPE T_Table_Routeur EST POINTEUR SUR T_Routeur_Cellule

TYPE T_Routeur_Cellule EST ENREGISTREMENT

    Adresse : T_Adresse_IP
    Masque : T_Adresse_IP
    Interface : Chaine de caracteres
    Suivant : T_Routeur_Cellule

FIN ENREGISTREMENT
```

R2 : Comment “Initialiser un routeur à partir d’un fichier texte”

```
Table_routage : T_Table_routage
routeur : T_Routeur
routeur <- Table_routage.all.routeur
Tant que ligne /= END_OF_LINE Faire          -- parcourir toutes les lignes
```

- Créer un tableau de taille 3 qui va stocker des chaînes de caractères (Destination, masque et interface)

ligne : in Chaine de caractère
tab : out TABLEAU (1..3) De Chaine de caractère

- Convertir l’adresse IP de la destination et du masque

tab(1) : in Chaine de caractère
destination : out T_Adresse_IP
tab(2) : in Chaine de caractère
masque : out T_Adresse_IP

- Enregistrer la destination et le masque une fois converti dans le routeur et enregistrer l’interface dans le routeur.

routeur : out T_Routeur
interface : in char
masque : in T_Adresse_IP
destination : in T_Adresse_IP

- Concevoir une nouvelle cellule dans le routeur pour stocker les données de la prochaine ligne.

routeur : out T_Routeur

FinTQ
{ ligne = END_OF_LINE }

R3 : Comment " Concevoir un tableau de taille 3 qui va stocker des chaînes de caractères(Destination , masque et interface) " ?

```
Donnee : TABLEAU (1..3) DE chaine de caractère
iterateur : ENTIER
iterateur ← 1
POUR j DE 1 A 3 PAS 1 FAIRE
    Donnee(j) = " "
    Tant que ligne(iterateur) /= ' ' Faire ( - - caractère différent d'un espace)
        Donnee(j) ← Donnee(j) & ligne(iterateur)
        iterateur ← iterateur + 1
    Fin TQ.
    iterateur ← iterateur + 1
Fin Pour.
```

R3 : Comment "Convertir l'adresse IP de la destination et du masque " ?

```
Adresse_converti = 0
destination = tab(1)
masque = tab(2)
N = length( destination )
POUR i DE 0 A 3 PAS 1 FAIRE (- - une adresse est de la forme 127.42.0.0)
    mot = ""
    j = 1
    caractere = destination(j)
    Tant que caractere /= '.' or j /= N Faire
        mot ← Concaténer(mot, caractere)
        i ← i+1
        caractere ← destination(i)
    Fin TQ
    nombre = Integer'Image(mot)
    Adresse_converti ← Adresse_converti + nombre * (2 ** (24-8*i))
Fin Pour
```

– – Faire de meme pour convertir le masque.

R3 : Comment " Enregistrer la destination et le masque une fois converti dans le routeur et enregistrer l'interface dans le routeur." ?

```
routeur.Adresse ← Adresse_converti
routeur.Masque ← Masque_converti
routeur.Interface ← tab(3) - - dans tab(3) se trouve l'interface.
```

R3 : Comment " Concevoir une nouvelle cellule dans le routeur pour stocker les données de la prochaine ligne." ?

```
routeur ← routeur.all.suivant
routeur ← new T_Routeur_Cellule
```

III.2 Cache LCA

III.3 Cache LA

R0 : Modéliser un routeur LA avec une politique LRU

R1 : Comment modéliser un routeur LA avec une politique LRU ?
TYPE T_Arbre_Cellule

TYPE T_Arbre EST POINTEUR SUR T_Arbre_Cellule

TYPE T_Cache_Arbre EST ENREGISTREMENT
Arbre : T_Arbre
Taille : Entier
Taille_Max : Entier
Defauts : Entier
Demandes : Entier
Enregistrement : Entier
Politique : T_Politique
FIN ENREGISTREMENT

TYPE T_Arbre_Cellule EST ENREGISTREMENT
Adresse : T_Adresse_IP
Masque : T_Adresse_IP
Sortie : Chaine de caractère
Gauche : T_Arbre
Droite : T_Arbre
Feuille : Booléen
Identifiant : Entier
Hauteur : Entier
FIN ENREGISTREMENT

R0 : Initialiser le cache

→ procédure Initialiser_Arbre Arbre : out T_Arbre

R1 : Comment initialiser le cache ?

Arbre ← Null Arbre : out T_Arbre

R0 : Initialiser les paramètres du cache

→ procédure Initialiser_Cache Cache : out T_Cache, Taille_Max : in Entier, Politique : in T_Politique

R1 : Comment initialiser les paramètres du cache ?

Cache.Taille_Max ← Taille_Max
Cache.Taille ← 0
Cache.Defauts ← 0
Cache.Demandes ← 0
Cache.Politique ← Politique

R0 : Savoir si l'arbre du cache est vide

→ fonction Est_Vide Arbre : in T_Arbre

R1 : Comment savoir si l'arbre du cache est vide ?

RETOURNE (Arbre = null)

R0 : Vider l'arbre du cache

→ procédure Vider Arbre : in out T_Arbre

R1 : Comment vider l'arbre du cache ?

Si non Est_Vide(Arbre) **Alors**
Appeler Vider(Arbre.ˆ.Gauche)
Appeler Vider(Arbre.ˆ.Droite)
Free(Arbre)
Sinon
Ne rien faire
Fin Si

R0 : Enregistrer dans l'arbre du cache les informations nécessaires

→ procédure Enregistrer Arbre : in out T_Arbre, Cache : in out T_Cache, Adresse : in T_Adresse_IP, Masque : in T_Adresse_IP, Sortie : in Unbounded_String, Politique : in T_Politique

R1 : Comment enregistrer dans l'arbre du cache les informations nécessaires ?

Traiter le cas où l'arbre du cache est vide Arbre : in T_Arbre
Récupérer la taille du masque Masque : in T_Adresse_IP, Taille_Masque : out Entier
Regarder chaque bit de l'adresse pour savoir si on est au niveau d'une feuille Taille_Masque : in Entier, Arbre : in out T_Arbre

R2 : Comment traiter le cas où l'arbre du cache est vide ?

Si Est_Vide(Arbre) **Alors**
Arbre ← NEW T_Arbre(0, 0, ıȷ, null, null, 0, False, 0, 0)
Sinon
Ne rien faire
Fin Si

R2 : Comment récupérer la taille du masque

Importer la fonction Get_Taille_Masque
Taille_Masque ← Get_Taille_Masque(Masque)

R2 : Comment regarder chaque bit de l'adresse pour savoir si on est au niveau d'une feuille ?

Si (Arbreˆ.Hauteur - Taille_Masque = 0) **Alors**
Stocker les informations nécessaires Adresse : in T_Adresse_IP, Masque : in T_Adresse_IP, Sortie : in Unbounded_String, Feuille : in Booléen

Mettre à jour les paramètres du cache selon sa politique Politique : in T_Politique, Cache : in out T_Cache

Sinon si (Adresse ET BINAIRE 2 ** (31 - Arbre^.Hauteur)) = 0 **Alors**

Regarder si le bit vaut 0 et si la cellule de gauche est vide Arbre : in out T_Arbre, Adresse : in T_Adresse_IP, Masque : in T_Adresse_IP, Sortie : in Unbounded_String, Politique : in T_Politique

Sinon

Regarder si le bit vaut 1 et si la cellule de droite est vide Arbre : in out T_Arbre, Adresse : in T_Adresse_IP, Masque : in T_Adresse_IP, Sortie : in Unbounded_String, Politique : in T_Politique

Fin Si

R3 : Comment stocker les informations nécessaires dans la feuille ?

Arbre.Adresse ← Adresse
Arbre.Masque ← Masque
Arbre.Sortie ← Sortie
Arbre.Feuille ← Vrai

R3 : Comment mettre à jour les paramètres du cache ? (LRU)

Arbre^.Identifiant ← 0
Cache.Taille ← Cache.Taille + 1
Cache.Enregistrement ← Cache.Enregistrement + 1

R3 : Comment regarder si le bit vaut 0 et si la cellule de gauche est vide ?

Si Est_Vide(Arbre^.Gauche) **Alors**
Enregistreur ← new T_Arbre_Cellule(0, 0, ıȷ, null, null, 0, Faux, 0, 0)
Enregistreur^.Hauteur ← Arbre^.Hauteur + 1
Arbre^.Gauche ← Enregistreur
Enregistreur ← null
Appeler Enregistrer(Arbre^.Gauche, Cache, Adresse, Masque, Sortie, Politique)
Sinon
Appeler Enregistrer(Arbre^.Gauche, Cache, Adresse, Masque, Sortie, Politique)
Fin Si

R3 : Comment se déplacer à droite car le bit vaut 1 ?

Si Est_Vide(Arbre^.Droite) **Alors**
Enregistreur ← new T_Arbre_Cellule(0, 0, ıȷ, null, null, 0, Faux, 0, 0)
Enregistreur^.Hauteur ← Arbre^.Hauteur + 1
Arbre^.Droite ← Enregistreur
Enregistreur ← null
Appeler Enregistrer(Arbre^.Droite, Cache, Adresse, Masque, Sortie, Politique)
Sinon
Appeler Enregistrer(Arbre^.Droite, Cache, Adresse, Masque, Sortie, Politique)
Fin Si

R0 : Supprimer une feuille de l'arbre

→ procédure Supprimer_LRU Arbre : in T_Arbre, Min_Identifiant : in Entier

R1 : Comment supprimer une feuille de l'arbre

Suppresseur ← Arbre
Traiter le cas de base Suppresseur : in T_Arbre
Appeler récursivement la procédure à gauche et à droite Suppresseur : in T_Arbre

R2 : Comment traiter le cas de base ?

Si non Est_Vide(Suppresseur) **Alors**
    Supprimer la cellule selon les critères de la politique Suppresseur : in T_Arbre, Min_Indentifiant : in Entier
Sinon
    Lever Suppression_Exception → Ne rien faire
Fin Si

R2 : Comment appeler récursivement la procédure à gauche et à droite ?

Appeler Supprimer_LRU(Suppresseur^.Gauche)
Appeler Supprimer_LRU(Suppresseur^.Droite)

R2 : Comment supprimer la cellule et mettre à jour la taille du cache

Free(Suppresseur)
Cache.Taille = Cache.Taille - 1

R3 : Comment supprimer la cellule selon les critères de la politique ?

Si Suppresseur^.Feuille **Et Alors** Min_Identifiant = Suppresseur^.Identifiant **Alors**
    Suppresseur^.Adresse ← 0
    Suppresseur^.Frequence ← 0
    Suppresseur^.Identifiant ← 0
    Suppresseur^.Masque ← 0
    Suppresseur^.Sortie ← ıȷ
    Suppresseur^.Feuille ← Faux
Sinon
    Ne rien faire
Fin Si

R0 : Afficher l'arbre du cache

→ procédure Afficher_Arbre Arbre : in T_Arbre

R1 : Comment afficher l'arbre du cache ?

Afficheur ← Arbre Afficheur : out T_Arbre
Regarder si le cache est vide et propager l'exception Afficheur : in T_Arbre
Regarder si on est au niveau d'une feuille et l'afficher Afficheur : in T_Arbre
Continuer par récursivité Afficheur : in T_Arbre

R2 : Comment regarder si le cache est vide et propager l'exception ?

Si Est_Vide(Afficheur) **Alors**
    Lever Affichage_Exception → Ne rien faire
Sinon
    Ne rien faire
Fin Si

R2 : Comment regarder si on est au niveau d'une feuille et l'afficher ?

Si Afficheur^.Feuille **Alors**
    Afficher("Adresse :" & T_Adresse_IP'Image(Afficheur.All.Adresse))
    Afficher("Masque :" & T_Adresse_IP'Image(Afficheur.All.Masque))
    Afficher(To_String(Afficheur.All.Sortie))
    Afficher("Identifiant :" & Integer'Image(Afficheur.All.Identifiant))
    Afficher("Fréquence :" & Integer'Image(Afficheur.All.Frequence))
Sinon
    Ne rien faire
Fin Si

R2 : Comment continuer par récursivité ?

Appeler Afficher_Arbre(Afficheur^.Gauche)
Appeler Afficher_Arbre(Afficheur^.Droite)

R0 : Afficher les statistiques relatives au cache et à sa politique

→ procédure Afficher_Statistiques_Cache Cache : in T_Cache

R1 : Comment afficher les statistiques relatives au cache et à sa politique ?

Afficher(Cache.Demandes)
Afficher(Cache.Defauts)
Taux_Defauts ← Réel(Cache.Demandes) / Réel(Cache.Defauts) Taux_Defauts : out Réel
Afficher(Taux_Defauts)

R0 : Chercher une adresse dans l'arbre

→ fonction Chercher_Arbre Arbre : in T_Arbre ; Cache : in out T_Cache ; Adresse : in T_Adresse_IP

R1 : Comment chercher une adresse dans l'arbre ?

Recherche_Adresse ← Arbre Recherche_Adresse : out T_Arbre
Compteur ← 1 Compteur : out Entier
Se déplacer jusqu'à l'adresse si elle existe Compteur : in Entier, Adresse : in T_Adresse_IP, Recherche_Adresse : in out T_Arbre
Traiter le cas où l'adresse n'est pas trouvée Compteur : in Entier, Adresse : in Entier, Cache : in out T_Cache, Recherche_Adresse : in T_Arbre
Traiter le cas où l'on est au niveau de l'adresse Sortie : out Unbounded_String, Recherche_Adresse : in out T_Arbre, Politique : in T_Politique

RETOURNE Sortie

R2 : Comment se déplacer jusqu'à l'adresse si elle existe ?

Tant Que Compteur /= 32 and Adresse /= Recherche_Adresse^.Adresse **Faire**
Si ((Adresse ET BINAIRE (2 ** (32 - Compteur))) = 0) **Alors**
Traiter le cas où le bit vaut 0 Adresse : in T_Adresse_IP, Recherche_Adresse : in out T_Arbre, Cache : in out T_Cache
Sinon
Traiter le cas où le bit vaut 1 Adresse : in T_Adresse_IP, Recherche_Adresse : in out T_Arbre, Cache : in out T_Cache
Fin Si
Compteur ← Compteur + 1
Fin Tant Que

R2 : Comment traiter le cas où l'adresse n'est pas trouvée ?

Si Compteur = 32 **Et** Adresse /= Recherche_Adresse^.Adresse **Alors**
Afficher("L'adresse" & T_Adresse_IP'Image(Adresse) & " n'a pas été trouvée")
Cache.Defauts ← Cache.Defauts + 1
Lever Adresse_Absente_Exception → RETOURNE To_Unbounded_String("L'adresse n'a pas été trouvée dans le cache.")
Sinon
Ne rien faire
Fin Si

R2 : Comment traiter le cas où l'on est au niveau de l'adresse ?

Sortie ← Recherche_Adresse^.Sortie
Recherche_Adresse^.Frequence ← Recherche_Adresse^.Frequence + 1
Si Politique = LRU **Alors**
Mettre à jour l'identifiant Recherche_Adresse : in out T_Arbre

Sinon
Ne rien faire
Fin Si

R3 : Comment traiter le cas où le bit vaut 0 ?

Si Est_Vide(Recherche_Adresse^.Gauche) **Alors**
Afficher("L'adresse" & T_Adresse_IP'Image(Adresse) & " n'a pas été trouvée")
Cache.defauts ← Cache.Defauts + 1
Lever Adresse_Absente_Eception → RETOURNE "L'adresse" & T_Adresse_IP'Image & " n'a pas été trouvée"
Sinon
Recherche_Adresse ← Recherche_Adresse^.Gauche
Fin Si

R3 : Comment traiter le cas où le bit vaut 1 ?

Si Est_Vide(Recherche_Adresse^.Droite) **Alors**
Afficher("L'adresse" & T_Adresse_IP'Image(Adresse) & " n'a pas été trouvée")
Cache.defauts ← Cache.Defauts + 1

Lever Adresse_Absente_Eception → RETOURNE "L'adresse" & T_Adresse_IP'Imag & " n'a pas été trouvée"
Sinon
Recherche_Adresse ← Recherche_Adresse^.Droite
Fin Si

R3 : Comment mettre à jour l'identifiant ?

Max ← Recherche_Identifiant_Max(Arbre) Max : out Entier
Si Max = 0 **Alors**
Recherche_Adresse^.Identifiant ← Recherche_Adresse^.Identifiant + 1
Sinon Si Recherche_Adresse^.Identifiant /= Max **Alors**
Recherche_Adresse^.Identifiant ← Recherche_Adresse^.Identifiant + 1
Sinon
Ne rien faire
Fin Si

R0 : Rechercher l'identifiant minimum

→ fonction Recherche_Identifiant_Min Arbre : in T_Arbre, Min : in out Entier

R1 : Comment rechercher l'identifiant minimum ?

Recherche_Min ← Arbre
Si non Est_Vide(Recherche_Min) **Alors**
Traiter le cas de base Min : in out Entier, Recherche_Max : in T_Arbre
Appeler récursivement la fonction Recherche_Identifiant_Min Min_Gauche : out Entier, Recherche_Min : in T_Arbre, Main : in Entier
Déduire le minimum en comparant le minimum à gauche et à droite Min : in out Entier, Min_Gauche : in Entier, Min_Droite : in Entier
Sinon
Lever Arbre_Vide_Exception → RETOURNE Min
Fin Si
RETOURNE Min

R2 : Comment traiter le cas de base ?

Si Recherche_Min^.Feuille **Et Alors** Recherche_Min^.Identifiant < Min **Alors**
Min ← Recherche_Min^.Identifiant
Sinon
Ne rien faire
Fin Si

R2 : Comment appeler récursivement la fonction Recherche_Identifiant_Min ?

Min_Gauche ← Recherche_Identifiant_Min(Recherche_Min^.Gauche, Min)
Min_Droite ← Recherche_Identifiant_Min(Recherche_Min^.Droite, Min)

R2 : Comment déduire le minimum en comparant le minimum à gauche et à droite ?

Si Min_Gauche > Min_Droite **Alors**

Min ← Min_Gauche
Sinon
Min ← Min_Droite
Fin Si

IV – Architecture et interdépendances

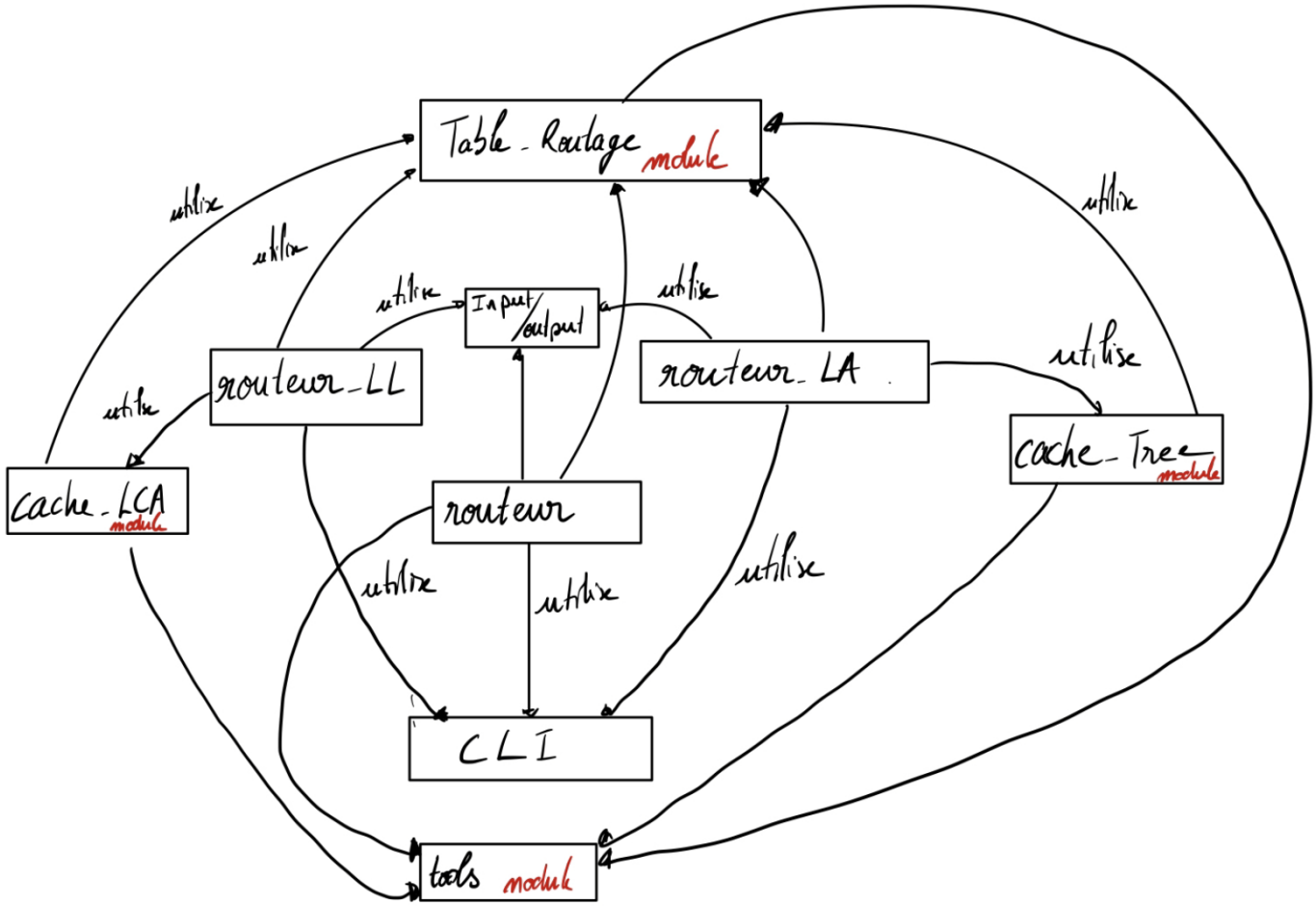


FIGURE IV.1 – l'architecture de l'application en modules

V – Méthodes des modules

V.1 Table de routage

Son rôle principal est d'initialiser une table de routage à partir d'un fichier texte (on pourra reprendre la structure des algorithmes pour initialiser un paquet à partir d'un fichier texte).

V.1.1 Types pour modéliser la table de routage

On utilisera une liste chaînée pour représenter la table de routage :

```
type T_Cellule;

type T_Table_Routage is access T_Cellule;

type T_Cellule is
    record
        Adresse : T_Adresse_IP;
        Masque : T_Adresse_IP;
        Sortie : Unbounded_String;
        Suivant : T_Table_Routage;
    end record;
```

On rappelle que les adresses sont de cette forme (codés sur 32 bits) : 124.11.145.0
En binaire on obtient : 1111100.00001011.10010011.0000000

On peut donc identifier les adresses de cette manière (de même pour les masques) :

```
type T_Adresse_IP is mod 2 ** 32;
```

V.1.2 Initialisation d'une table de routage

Le principal algorithme est :

```
Initialiser(param : in T_Param; Table_Routage: out T_Table_Routage)
```

(param : va contenir le fichier texte de notre table de routage)
L'algorithme fonctionne ainsi :

- Lire le fichier texte ligne par ligne.

- Traiter chaque ligne en la distinguant en trois partis : Adresse, Masque, Interface (ou Sortie).

- Convertir l'adresse et le masque qui sont des chaînes de caractères en :

T_Adresse_IP

- Stocker les données dans la table de routage

V.1.3 Exemple : Initialisation d'une table de routage

Voici le contenu du fichier table.txt contenant une table de routage et sa représentation simplifié (sans conversion des adresses et masques) :

```
147.127.16.0  255.255.240.0  eth0
147.127.18.0  255.255.255.0  eth1
147.127.0.0   255.255.255.0  eth2
212.0.0.0     255.0.0.0      eth3
0.0.0.0       0.0.0.0        eth0
```

FIGURE V.1 – fichier texte table de routage

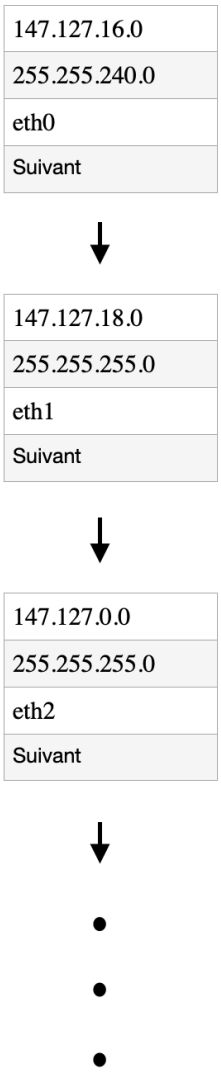


FIGURE V.2 – Illustration de la table de routage

V.2 Utilisation de la table de routage

La table de routage a pour but de stocker des adresses IP et de rediriger les requêtes en fonction de sa base de donnée. Une requête se fait à l'aide d'un paquet, qui est en vérité une adresse IP aussi.

Donc comment router un paquet ?

V.2.1 Routage d'un paquet

Le routage d'un paquet se procède comme suit :

V.2.1.1 Parcours de la table de routage

On parcourt la table de routage adresse par adresse. L'objectif est de trouver le meilleur chemin (appelée Cellule dans notre structure de donnée) éligible pour l'adresse, c'est à dire le chemin avec le plus grand masque.

V.2.1.2 Quand est-ce qu'un chemin est éligible ?

Un chemin est éligible si l'adresse destination, appliquée au masque du chemin, correspond à l'adresse du chemin.

V.2.1.3 Exemple

Prenons en exemple la table de routage V.1 affichée précédemment.
Soit ADRESSE = 147.127.16.7 on veut router cette adresse.

1. Chemin : 147.127.16.0 - 255.255.240.0 - eth0
   On applique ADRESSE au masque : c'est une opération logique ET bit à bit entre ADRESSE et le masque
   $$147.127.16.7 \quad ET \quad 255.255.240.0 = 147.127.16.0$$
   Le résultat correspond bien à l'adresse de ce chemin. Le chemin est donc éligible
2. Chemin : 147.127.18.0 - 255.255.255.0 - eth1
   $$147.127.16.7 \quad ET \quad 255.255.255.0 = 147.127.16.0$$
   Le résultat ne correspond pas à l'adresse de ce chemin 147.127.18.0. Ce chemin n'est pas éligible
3. Chemin : 147.127.0.0 - 255.255.0.0 - eth2
   $$147.127.16.7 \quad ET \quad 255.255.0.0 = 147.127.0.0$$
   Le résultat correspond bien à l'adresse de ce chemin. Ce chemin est donc éligible
4. Chemin : 212.0.0.0 - 255.0.0.0 - eth3
   $$147.127.16.7 \quad ET \quad 255.0.0.0 = 147.0.0.0$$
   Le résultat ne correspond pas à l'adresse de ce chemin 147.127.18.0
5. Chemin : 0.0.0.0 - 0.0.0.0 - eth0
   $$147.127.16.7 \quad ET \quad 0.0.0.0 = 0.0.0.0$$
   Le résultat correspond bien à l'adresse de ce chemin. Ce chemin est donc éligible

On a donc vu que 3 chemins de la table de routage sont éligibles. Mais quel chemin est le "meilleur" ? Comme dit précédemment, on va prendre celui qui possède le masque le plus long. En effet, plus le masque est long, moins celui-ci "masque" l'adresse IP, et donc plus l'adresse du chemin "ressemble" à l'adresse routée.

C'est pourquoi le chemin de d'adresse 0.0.0.0 et de masque 0.0.0.0 est appelé "Chemin par défaut", car n'importe quelle adresse peut prendre ce chemin.

V.3 Cache de type liste-chainée

V.4 Initialisation du cache LCA

V.4.1 Types pour modéliser le cache LA

On utilisera une liste chaînée pour stocker les adresses qu'on souhaite sauvegarder dans le cache. Les différents types utilisés sont :

```
type T_Cellule ;

type T_CACHE_LCA is access T_Cellule ;

type T_Cellule is record
    Adresse : T_ADRESSE_IP ;
    Masque : T_ADRESSE_IP ;
    Eth : Unbounded_String ;
    Frequence : Integer ;
    Suivant : T_CACHE_LCA ;
end record ;

end CACHE_LCA ;
Type T_CACHE_LCA is limited private
```

V.4.2 Utilisation du cache LCA

Le cache possèdera les fonctionnalités classiques d'une liste chaînée.

Il faudra ainsi initialiser le cache, y enregistrer des valeurs dans les différentes cellules associées, et enfin vider le cache entièrement une fois utilisé.

V.4.3 Politique de suppression du cache LCA

Une fois le cache plein, il faut supprimer une adresse au cache. Trois politiques vont être utilisées ici, la politique FIFO (First In First Out), la politique LRU (Less Recently Used) et la politique LFU (Less Frequently Used).

V.4.4 FIFO

La politique FIFO va donc consister à supprimer l'adresse qui a été ajoutée la première dans le cache. Pour cela, on va simplement supprimer le premier élément de la liste chaînée, puisque le premier élément ajouté à une liste chaînée se trouve être au début de la liste.

V.4.5 LRU

La politique LRU quant à elle va consister à supprimer l'élément le moins récemment utilisé. Pour ce faire, on a créé un tableau d'adresse recensant dans l'ordre les adresses ajoutées au cache. De plus, quand une adresse est réutilisée, on va la supprimer de ce tableau, et la remettre à la fin de ce même tableau, ainsi, pour supprimer l'adresse souhaitée, il suffira de supprimer le premier élément de ce

tableau nommé RECENT_LCA.

V.4.6 LFU

La politique LFU va consister à supprimer l'adresse la moins fréquemment utilisée. Pour la trouver, on va ajouter à chaque adresse une fréquence qu'on va incrémenter de 1 dès qu'on réutilise cette même adresse.

V.5 Cache de type Arbre

V.6 Initialisation du cache LA

Le rôle du cache LA est de stocker les routes les plus utilisées afin d'optimiser la recherche d'une destination en plus de la table de routage.

V.6.1 Types pour modéliser le cache LA

On utilisera un arbre préfixe pour représenter la structure du cache et un enregistrement pour stocker les paramètres du cache :

```
type T_Arbre_Cellule ;

type T_Arbre is access T_Arbre_Cellule ;

type T_Arbre_Cellule is record
    Adresse : T_Adresse_IP ;
    Masque : T_Adresse_IP ;
    Sortie : Unbounded_String ;
    Gauche : T_Arbre ;
    Droite : T_Arbre ;
    Frequence : Integer ; – LFU
    Feuille : Boolean ;
    Identifiant : Integer ; – LRU et FIFO
    Hauteur : Integer ;
end record ;

type T_Cache is record
    Taille : Integer ;
    Taille_Max : Integer ;
    Defauts : Integer ;
    Demandes : Integer ;
    Enregistrement : Integer ; – nombre d'enregistrement
    Politique : T_Politique ;
end record ;
```

V.6.2 Initialisation du cache LA

On utilise deux procédures distinctes pour initialiser le cache. Une pour les paramètres du cache et l'autre pour son arbre :

```
Initialiser_Arbre(Arbre : out T_Arbre)
Initialiser_Cache(Cache : out T_Cache ; Taille_Max : in Integer ; Politique : in T_Politique)
```

La procédure qui initialise l'arbre va initialiser le pointeur sur null. Celle initialisant les paramètres du cache va mettre la capacité du cache (Taille_Max) à la taille rentrée par l'utilisateur. Elle va aussi stockée la politique renseignée par l'utilisateur. Enfin, elle mettra l'ensemble des paramètres restant égaux à 0.

V.7 Utilisation du cache LA

Le cache a pour but de stocker des routes constituées d'une adresse IP, d'un masque et d'une interface de sortie. Elle permet l'optimisation de la recherche d'une destination car elle va stockée les routes les plus pertinentes selon la politique. En effet, cela peut être les routes les plus utilisées ou bien les plus récemment utilisées.

Ainsi, je vais présenter dans la suite les différentes procédures et fonctions permettant le fonctionnement du cache LA.

V.7.1 Savoir si le cache est vide

La procédure permet de savoir si le cache est vide ou non. Elle renvoie alors un booléen vrai si le cache est vide. Ainsi, il suffit de tester si l'arbre pointe sur null.

V.7.2 Vider le cache

La procédure permet de vider le cache en profondeur grâce à la récursivité. Le fonctionnement est simple, si le cache est vide, il n'y a rien à faire. Sinon, il faut appliquer la procédure Vider sur le fils gauche de l'arbre puis sur son fils droit. Enfin, il faut désallouer la cellule courante.

V.7.3 Récupérer les paramètres du cache ou de l'arbre

Il est nécessaire de coder des fonctions permettant de récupérer les paramètres du cache ou de l'arbre à cause de l'encapsulation. Les fonctions sont simples et se limitent au renvoi du paramètre dont il est question.

Les fonctions que j'ai implémenté sont les suivantes :

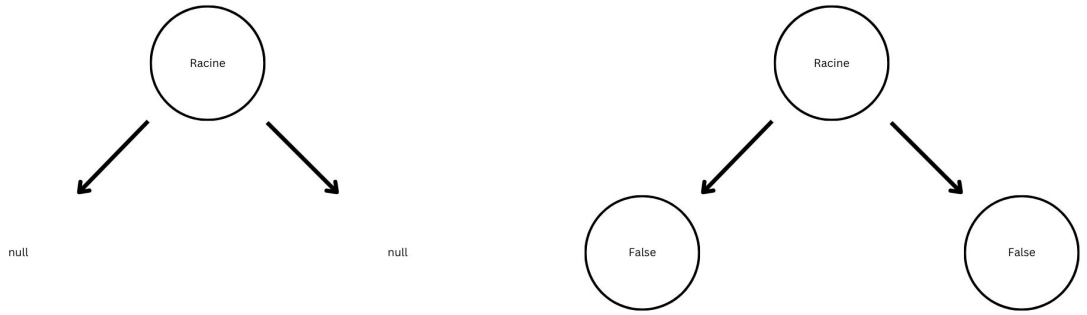
1. function Taille_Cache(Cache : in T_Cache) return Integer
2. function Frequence_Arbre(Arbre : in T_Arbre) return Integer
3. function Demandes_Cache(Cache : in T_Cache) return Integer
4. function Defauts_Cache(Cache : in T_Cache) return Integer
5. function Enregistrement_Cache(Cache : in T_Cache) return Integer

V.7.4 Enregistrer dans le cache

L'idée est d'enregistrer dans le cache une route constituée d'une adresse IP, d'un masque et d'une interface de sortie. D'autre part, il faudra modifier l'arbre ainsi que les paramètres du cache selon la politique.

On peut réaliser cette procédure de manière récursive. Ainsi, le cas de base consiste à regarder si l'arbre est vide et à créer une cellule en conséquence. Cette cellule sera initialisée avec des valeurs par défaut. On s'assurera seulement de mettre sa variable 'Feuille' à 'False' pour préciser que cette cellule ne stocke aucune information importante. Les figures V.3a et V.3b illustrent cela.

Ensuite, il faut traiter différents cas.



(a) Avant (b) Après

FIGURE V.3 – Arbre avant et après le cas de base.

Le premier consiste à regarder si on est au niveau d'une feuille. Pour cela, on peut regarder si la différence entre la hauteur de l'arbre et la taille du masque est nulle. Si c'est le cas, cela veut dire que la route doit être stockée à cette hauteur. Ainsi, on va stocker l'adresse IP, le masque et l'interface de sortie. On veillera de même à mettre 'Feuille' sur True pour indiquer que c'est une feuille.
Par suite, selon la politique, on initialise l'identifiant de la cellule. Dans le cas de la politique FIFO, l'identifiant sera mis au nombre d'enregistrement dans le cache. Cela a pour intérêt de connaître l'historique d'enregistrement des cellules. On fait la même chose pour la politique LFU car des cellules pourront avoir la même fréquence. En effet, cela est le cas si l'on utilise autant de fois deux mêmes adresses IP. Ainsi, dans ce cas, je supprimerai la route qui a été ajoutée en première. Il s'agit d'une combinaison de la politique LFU et FIFO. Ainsi, on se ramènera à ne supprimer qu'une seule adresse dans tous les cas. Pour l'autre politique, l'identifiant sera initialisé à 0. Enfin, il faut s'assurer d'augmenter la taille du cache dans ses paramètres et son nombre d'enregistrement de 1.
L'autre cas consiste à naviguer jusqu'à l'adresse en question. Pour cela, on regarde chaque bit de l'adresse pour savoir si il vaut 0 ou 1. On va à gauche si le bit vaut 0 et à droite si le bit vaut 1. Par exemple, le bit de poids fort de l'adresse vaut 0 si (Adresse AND 2 ** (31 - Arbre.All.Hauteur)) = 0. La hauteur de l'arbre étant incrémenté à chaque appel récursif, cela permet de descendre dans la structure de l'arbre. Ensuite, il faut regarder si la cellule en question est nulle ou non. Si la cellule est nulle, on initialise un pointeur temporaire appelé 'Enregistreur' et on initialise une nouvelle cellule comme dans le cas de base. Il s'agit ensuite d'incrémenter la hauteur de l'arbre via ce pointeur temporaire Enregistreur.All.Hauteur := Arbre.All.Hauteur + 1. Enfin, on raccroche l'arbre à cette nouvelle cellule Arbre.All.Gauche := Enregistreur puis on fait pointer "Enregistreur" sur null. Il ne reste plus qu'à faire un appel récursif de la procédure Enregistrer sur le fils gauche de l'arbre si le bit vaut 0 ou sur le fils droite de l'arbre si le bit vaut 1. La figure V.4 montre l'état de l'arbre après enregistrement.

V.7.5 Suppression d'un chemin du cache

La suppression dans le cache a lieu lorsque ce dernier est plein. La suppression s'effectue de manière différente selon la politique utilisée. On peut alors considérer une procédure 'Supprimer' générale qui redirige vers des sous-procédures qui agissent différement selon la politique. On s'assurera de diminuer la taille du cache de 1 à la fin de l'éxécution de cette procédure. De même, on s'assurera de récupérer l'adresse et l'identifiant minimum selon la politique utilisée.

V.7.5.1 FIFO

Pour supprimer le premier élément enregistré dans l'arbre. Il suffit que je fasse une recherche de l'identifiant minimum. Une fois que cela est fait, je vais faire un parcours préfixe dans l'arbre afin de trouver la cellule qui correspond. Ainsi, je peux résoudre le problème de manière récursive.

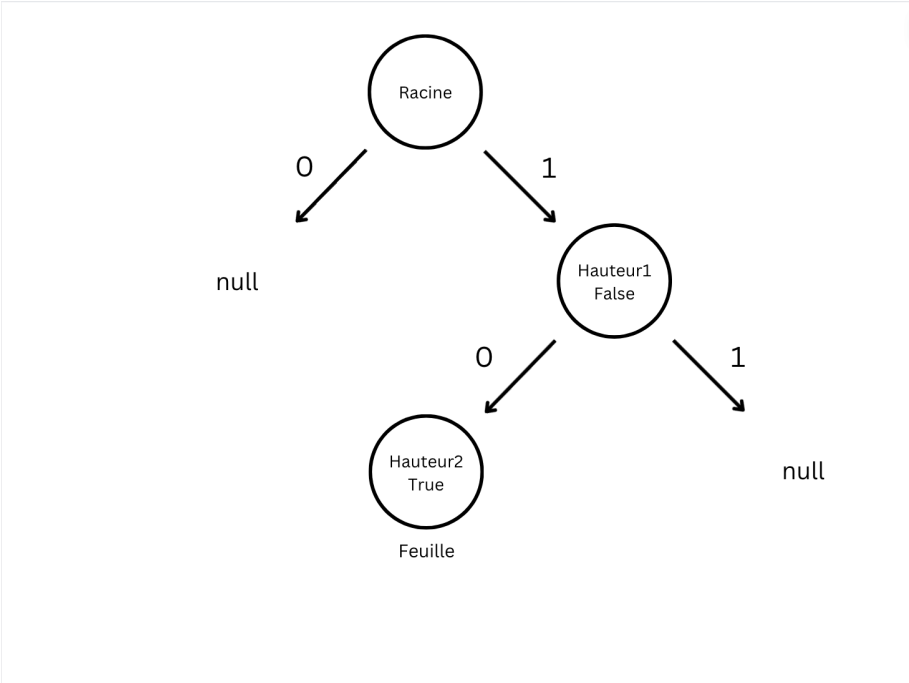
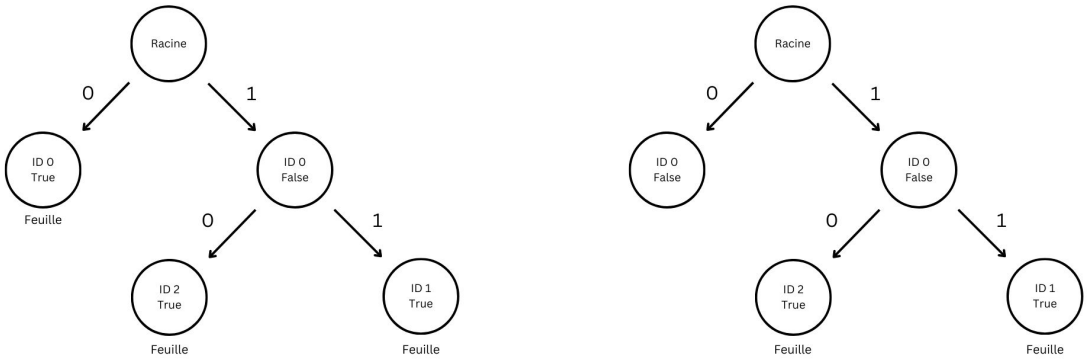


FIGURE V.4 – Arbre après enregistrement.



(a) Avant (b) Après

FIGURE V.5 – Arbre avant et après la suppression de manière générale.

On commence par regarder si l'arbre est 'null'. Dans ce cas, je propage l'exception sur l'instruction 'null'. Par suite, je regarde le cas si l'adresse correspond tout en s'assurant d'être au niveau d'une feuille. Il ne reste plus qu'à réinitialiser la cellule et passer la variable 'Feuille' à 'False'. J'appelle ensuite cette procédure 'Supprimer_FIFO' récursivement sur le fils gauche et le fils droit. Les figures V.5a et V.5b montrent l'état du cache avant et après la suppression selon la politique FIFO mais le schéma se généralise aux autres politiques.

V.7.5.2 LRU

On procède de la même manière que pour la suppression avec la politique FIFO. La différence réside dans la manière d'attribuer l'identifiant à la cellule. Ceci est expliqué dans la partie V.7.12 concernant la recherche d'une interface de sortie dans le cache à partir d'une adresse IP.

V.7.5.3 LFU

Cette fois, il s'agit de chercher la fréquence minimale. En effet, à chaque fois que l'on va utiliser une route dans le cache, on va incrémenter le paramètre 'Fréquence' de la cellule. Il s'agira ensuite de faire une recherche de la fréquence minimale. La fonction sera détaillée ci-dessous. La suite est similaire à la méthode de suppression utilisée pour les autres politiques. D'autre part, le cas où deux cellules ont la même fréquence est expliquée dans la partie V.7.4 d'enregistrement dans le cache.

V.7.6 Rechercher un identifiant ou une fréquence minimale dans le cache

Pour rechercher un identifiant ou une fréquence minimale dans le cache, on peut initialiser un pointeur 'Recherche_Min' sur 'Arbre'. Ensuite, on traite le problème de manière récursive en effectuant un parcours préfixe sur l'arbre. Il s'agit alors de regarder dans un premier temps si le cache est vide ou non. Si la cellule est vide alors on lève une exception 'Arbre_Vide_Exception' qui se propage en retournant le minimum. Si la cellule n'est pas vide alors on traite le cas de base. Le cas de base consiste à regarder si la cellule est une feuille puis si l'identifiant est plus petit que le minimum actuel. On appelle ensuite récursivement la fonction sur le fils gauche et le fils droit. Enfin, on compare le minimum de l'arbre gauche et de l'arbre droit. Il s'agit alors de retourner le plus petit des deux.

V.7.7 Rechercher l'identifiant minimal pour une fréquence donnée dans le cache

Il s'agit de la même démarche que dans la partie V.7.6. Il suffit de rajouter une condition sur le cas de base, où l'identifiant est plus petit que le minimum actuel. En effet, il faut aussi s'assurer que la fréquence de la cellule correspond à la fréquence donnée. On peut alors affecter le minimum à l'identifiant de la cellule.

V.7.8 Rechercher l'identifiant maximum dans le cache

Il s'agit de la même démarche que dans la partie V.7.6 sauf qu'il s'agit ici de regarder si l'identifiant est plus grand que le maximum actuel.

V.7.9 Savoir si le cache est plein

Il suffit de regarder si la taille du cache (Cache.Taille) est supérieure ou égale à la capicité du cache (Cache.Taille_Max). Si c'est le cas, la fonction renvoie True sinon elle renvoie False.

V.7.10 Afficher le cache

Pour afficher le cache, il faut faire un parcours en profondeur. Pour cela, on peut coder cette procédure récursivement. On peut commencer par initialiser un pointeur temporaire sur Arbre qui s'appelle "Afficheur". Le cas de base correspond à un arbre vide. Ainsi, on lève une exception qui se propage sur l'instruction "null". Ensuite, on peut regarder si on est au niveau d'une feuille Afficheur.All.Feuille. Si c'est le cas, on affiche l'adresse, le masque, la sortie et éventuellement d'autres informations intéressantes. Enfin, il reste à appeler la procédure d'affichage sur le fils gauche de l'arbre puis sur le fils droit.

V.7.11 Afficher les statistiques du cache

Pour afficher les statistiques du cache, il suffit de récupérer les paramètres du cache et de les afficher. On veillera cependant à convertir le nombre de demandes et de défauts du cache en réel pour calculer le taux de défauts.

V.7.12 Chercher une interface de sortie dans l'arbre

La dernière procédure consiste à chercher une interface de sortie dans l'arbre à partir d'une adresse IP donnée. L'idée est d'initialiser un pointeur temporaire "Recherche_Adresse" sur l'arbre. Ensuite, on

se déplace jusqu'à l'adresse en utilisant une boucle while. Les conditions à satisfaire sont les suivantes : Compteur /= 32 and Adresse /= Recherche_Adresse.All.Adresse. Ainsi, soit l'adresse correspond et l'on se retrouve au niveau d'une feuille, soit on ne trouve pas l'adresse.

Pour se déplacer jusqu'à l'adresse, on compare chaque bit de l'adresse pour savoir si il vaut 0 ou 1. On utilise le compteur pour parcourir les bits de l'adresse. Dans le cas où l'on tombe sur un pointeur null, cela veut dire que l'adresse n'existe pas et on lève une exception qui correspond à ce cas. De même, si l'on sort de la boucle car le compteur vaut 32 et que l'adresse ne correspond pas alors on lève cette même exception. Sinon, cela veut dire que l'adresse correpond. Dans ce cas, on récupère la sortie sur la cellule en question.

De plus, si la politique est LRU, on recherche l'identifiant maximum dans l'arbre. Si le maximum vaut 0 alors on augmente l'identifiant de la cellule à 1 car cela veut dire qu'on utilise le cache pour la première fois. Sinon, si l'identifiant de la cellule est différent du maximum alors on affecte l'identifiant au maximum que l'on somme de 1 (Recherche_Adresse.All.Identifiant := Max + 1). De cette manière, on s'assure d'avoir en mémoire la cellule la plus récemment utilisée.

On finit alors par renvoyer l'interface de sortie.

V.8 Test

Les tests ont permis de déceler de nombreuses erreurs dans les programmes
Les tests consistaient (selon la procédure testée) à utiliser un petit échantillon (adresse, masque, sortie) et voir si tout fonctionnait bien.
On pouvait également programmer des procédures qui affichaient le contenu de nos caches ou table de routage.

```
procedure Test_Enregistrer is
   Cache : T_CACHE_LCA;
   Adresse : T_Adresse_IP;
   Masque : T_Adresse_IP;
   Eth : Unbounded_String;
begin
   -- Initialisation de la route @ mettre en cache
   Initialiser(Cache, 2, LFU);
   Adresse := Unbounded_String_To_Adresse_IP(To_Unbounded_String("192.168.0.0"));
   Masque := Unbounded_String_To_Adresse_IP(To_Unbounded_String("255.255.0.0"));
   Eth := To_Unbounded_String("eth0");

   -- Mise en cache de la route pr@c@demment cr@@e
   Enregistrer(Cache, Adresse, Masque, Eth);

   pragma Assert(Adresse_Presente(Cache, Adresse));
   pragma Assert(Recuperer_Masque_Cache(Cache, Adresse) = Masque);
   pragma Assert(Recuperer_Eth_Cache(Cache, Adresse) = Eth);
```

FIGURE V.6 – Exemple de test

VI – Conclusion

Le projet est fonctionnel dans son ensemble : trois programmes principaux sont disponibles pour essayer les diverses faces de notre travail : routeur, simple sans cache ; routeur_L, utilisant un cache sous forme de liste chainée ; et routeur_LA, utilisant un cache sous forme d'un arbre binaire.

Malheureusement, l'affichage statistique du cache_LCA ainsi que l'option "affichage des paramètres" n'ont pas pu être implémenté dans les temps.

Les perspectives d'améliorations sont nombreuses : Tout d'abord, nous souhaiterons regrouper les deux modules caches via un module générique "Cache" prenant en paramètre la politique, la taille du cache, et surtout le type abstrait de donnée qui stockera les différents chemins du cache.

Ensuite, comme précisé par la suite dans le rapport, plusieurs fonctions et procédures pourraient être optimisées afin de gagner en temps d'exécution. Le projet étant très peu coûteux, c'est peu gênant.

Jérémy La principale difficulté de ce projet pour moi fût le rapport en LateX. J'ai déjà acquis de solides bases en programmation durant mes années en licence informatique, et j'ai pu ainsi aider mes collègues lorsqu'ils ne comprenaient pas leur erreur de compilation. J'ai beaucoup aimé ce projet car il demande beaucoup d'algorithmes différents assez difficile si on les veut parfait.

Julien J'ai appris à écrire un rapport en Latex (C'est très stylé le Latex), et au niveau de la programmation du module table de routage il n'y a pas eu trop de diffculté comme c'était assez similaire à ce qu'on avait déjà fait en TP, TD, mini projet (et aussi parceque Jeremy ce beau gosse m'a beaucoup aidé)

Alan Je me suis occupé du cache LCA . J'ai eu une première expérience sur le travail en commun sur un même projet, ce qui m'a permis d'apprendre à utiliser GitHub notamment.

Alexandre Je me suis chargé du cache modélisé par un arbre préfixe. Les procédures du module sont fonctionnelles cependant il y a des optimisations à apporter. En effet, lorsque deux adresses IP sont similaires mais que l'une est plus longue que l'autre alors l'adresse IP la plus courte n'est plus vraiment au niveau d'une feuille. D'autre part, ma procédure 'Supprimer' ne désalloue pas vraiment la cellule mais la masque d'une certaine manière. Ainsi, je ne compte dans la taille que les feuilles du cache. Cependant, si je prenais comme taille du cache l'ensemble des noeuds alors la procédure 'Supprimer' actuelle viendrait poser de gros problèmes d'optimisation dans la gestion de la taille du cache. Néanmoins, ce projet m'a permis de mettre mes compétences de programmation à l'épreuve et de réussir à faire fonctionner le cache d'une certaine manière. Je pense y avoir passé plus d'une vingtaine d'heures sans compter le temps de comprendre le sujet auparavant. D'autre part, l'implémentation reste perfectible. Certaines procédures/fonctions peuvent user de la surchage et devenir générique. D'autres réalisent presquent la même chose. De plus, pour supprimer un élément du cache, je refais un parcours préfixe à chaque fois. L'utilité d'utiliser un arbre préfixe est de pouvoir se déplacer rapidement vers une cellule en connaissant le chemin jusqu'à celle-ci. Ainsi, je ne pense pas avoir suffisamment exploiter cela. Je pense qu'il y a moyen d'utiliser cela. Au niveau des types, j'aurais pu user davantage de l'encapsulation. Finalement, ce projet m'a été très formateur et m'a permis de me confronter aux concepts vus lors des cours de programmation impérative.

A – Bibliographie

[1] Site Internet de l'école d'ingénieur ENSEEIHT

[2] Page wikipédia sur la définition d'un arbre binaire

[3] Notre enseignant de TP