Rapport de projet

Routage

Mathis Sigier, Benoit Maggion, Aléxy Fièvet

Programmation Impérative

ENSEEIHT

Janvier 2023

Plan du rapport

1. Introduction du sujet
2. Raffinages
3. Architecture générale du code final

I - Introduction

Le routage en informatique est un processus d'acheminement de données à travers un réseau vers une interface liée aux informations données. C'est la table de routage qui structure l'ensemble des informations pour l'acheminement. C'est une façon efficace et rapide d'envoyer un grand nombre d'informations vers des interfaces choisies.

Dans le cadre de notre enseignement de la matière programmation impérative. Nous avons eu à réaliser un code complet et efficace de l'utilisation d'un routeur, d'une table de routage et du cache associé.

Nous avons utilisé le language Ada pour écrire notre code.

Nous avons un mois et demi pour réaliser ce projet en trio.

Après avoir pris connaissance du sujet, nous avons rapidement commencé à écrire le raffinage du projet pour pouvoir avoir une vision d'ensemble du code que nous allions créer avec Ada. Les raffinages constituent donc une base que nous avons suivi tout au long de la rédaction des différents codes.

Les objectifs généraux du projet étaient de pouvoir extraire la table de routage inscrite dans un fichier texte pour l'implémenter dans une LCA, être capable de traiter les différentes destinations des paquets qui étaient aussi inscrites dans un fichier texte et générer un fichier résultat.

Dans un premier temps nous avons codé un routeur simple ne fonctionnant sans cache. Puis nous avons coder deux routeurs utilisant des caches : un routeur disposant d'un cache sous forme de liste chainée, l'autre utilisant un arbre.

II - Raffinages

Routeur Simple :

R0 : Afficher la bonne destination pour un paquet reçu.

R1 : **Comment** : "Afficher la bonne destination pour un paquet reçu." ?

- Mettre la table de routage sous forme de liste chainée.
- Choisir la destination la plus adaptée.
- Afficher cette destination. destination_finale : out.

R2 : **Comment** : "Mettre la table de routage sous forme de liste chainée."?

- Classer les valeurs de la table par type. destination; masque; interface : in
- Convertir l'adresse de la destination en binaire. destination_binaire : out.
- Convertir le masque en binaire. masque_binaire : out.
- T_Table de type Pointeur sur liste_chainée T_route.

R3 : Comment : "Table de type Pointeur sur liste_chainée route." ?

Type T_route est :
    destination_binaire : mod 2**32
    masque_binaire : mod 2**32
    interface : string
    suivant : T_route

R2 : **Comment** : "Choisir la destination la plus adaptée." ?

- Recupérer l'adresse IP du paquet. adresse : in.
- Convertir l'adresse IP du paquet en binaire. adresse_binaire : out.
- Sélectionner la route la plus adaptée.

R3 : **Comment** : "Sélectionner la route la plus adaptée." ?

- Classer les routes en fonction de la taille des masques (du plus long au plus court.) taille_masque_i : out.
- Sélectionner la première route correspondant (avec le bon nombre de bits à comparer (représenté par la taille du masque)). destination_finale : in.

R4 : **Comment** : "Classer les routes en fonction de la taille des masques." ?

Pour i allant de 1 à nombre_masque **faire:**
reste = 0
masque2 = masque
taille_masque_i = 32
Tant que reste = 0 et taille_masque_i >0 **Faire**
reste = masque2%2;

masque2 = masque2/2
taille_masque_i = taille_masque_i - 1
Fin Tant Que
Fin Pour

R4 : **Comment** : "Sélectionner la première route correspondant." ?
Pour i allant de 1 à nombre_route **faire**
Comparer l'adresse IP avec la destination.

R5 : **Comment** : "Comparer l'adresse IP avec la destination." ?
masque2 = masque
destination2 = destination
Pour j allant de 1 à taille_masque **faire**
masque2 = masque2/2
destination2 = destination2/2
Si masque2 = destination2 **faire**
interface_finale = interface
renvoyer interface_finale
Sinon faire
renvoyer destination_par_defaut
Fin Si

Détails du cache :

Routeur LL avec les trois types de politiques:

R0: Utiliser un routeur LL qui propose les trois politiques FIFO, LRU et LFU

R1: Comment "Utiliser un routeur LL qui propose les trois politiques FIFO, LRU et LFU"?

Initialiser un routeur LL qui propose les trois politiques FIFO, LRU et LFU (Taille_cache : Integer, Politique_choisie : String, Statistique : Boolean, Table : Text_file, Paquets : Text_file, Résultats : Text_file, TableR : LC, SD_Cache : LC): **out**

Traiter les paquets Paquets : **in out**

Restituer les résultats Resultats : **in out**

R2: Comment "Initialiser un routeur LL qui propose les trois politiques FIFO, LRU et LFU"?

Récupérer les caractérisitiques, la politique et les données (Taille_cache : Integer, Politique_choisie : String, Statistique : Boolean, Table : Text_file, Paquets : Text_file, Résultats : Text_file): **out**

Traiter l'ensemble des commandes reçues (Taille_cache : Integer, Politique_choisie : String, Statistique : Boolean, Table : Text_file, Paquets : Text_file, Résultats : Text_file): **in**
Initialiser la table de routage sous forme de liste chaînée (TableR) : **out**

Initialiser la structure de données qui représente le cache (SD_cache) : **out**

R2 : Comment "Traiter les paquets"?
Pour i **dans** 1..Nombre_paquets **Faire** - - On suppose que l'on connait le nombre total de paquets dans le fichier Paquets
Traiter le paquet_i Paquet_i : **in**
FinPour

R2: : Comment "Restituer les résultats"?
Pour i **dans** 1..Nombre_paquets **Faire**
Ecrire(Str(Paquet_i)) - - On affiche le paquet demandé et l'interface de destination sur la même ligne
Ecrire(Resultats(i)) - - Affiche l'interface de sortie
Retour à la ligne
FinPour
Si Statistique = True **Faire**
Afficher les statistiques
Fin Si

R3 : Comment "Traiter le paquet_i"?
Choisir la destination la plus adaptée dans le cache (Route_cache : Boolean, Destination_cache : String): **out,** (Paquet_i) : **in** - - Destination cache est l'interface d'arrivée
Si Route_cache = False **faire** - - Pas de destination trouvée dans le cache
Choisir la destination la plus adaptée dans la table de routage (Paquet_i :) **in**

Fin Si
Mettre à jour le cache en fonction de la politique choisie (Paquet_i): **in, (**SD_cache) : **in out**

R3: Comment "Récupérer les caractéristiques, la politique et les données" ?
Taille_cache <- 10 - - On met dans un premier temps toutes les valeurs par défauts
Politique_choisie <- "FIFO"
Statistique <- '-s'
Table <- table.txt
Paquets <- paquets.txt
Resultats <- resultats.txt

R3: Comment "Traiter l'ensemble des commandes reçues"?
Créer un tableau recensant l'ensemble des commandes (Tableau_commandes) : **out,**
Numero_argument <- 1
Tant que Numero_argument < Nombre_argument **Faire** - - Il y a maximum 5 arguments lors de l'appel de la fonction, on suppose Nombre_argument connu
Traiter l'argument reçu Numero_argument : **in**

Numero_argument <- Numero_argument +1
FinTantque

R3: Comment initialiser la table de routage sous forme de liste chaînée?
Initialiser (TableR) - - On initialise une liste chaînée vide (programme équivalent lors du mini projet 2)
Pour i dans 1..Nb_lignes_Table **Faire** - - On suppose que l'on connaît le nombre de lignes total du fichiers texte table
Enregistrer_Ligne_i
FinPour

R3: Comment "Initialiser la structure de données qui repéresente le cache"?
Initialiser(StructD_Cache)

R4 : Comment "Mettre à jour le cache en fonction de la politique choisie"?
Si Taille(SD_Cache)>=Taille_cache **Faire**
Selon Politique_choisie **Dans**
"FIFO" => Supprimer Donnee_la_plus_ancienne(SD_cache)
"LRU" => Supprimer Donnee_moins_recemment_utilisee(SD_cache)
"TFU" => Supprimer Donnee_moins_utilisee(SD_cache)

FinSelon
FinSi
Implanter dans le cache le dernier paquet utilisé

R4: Comment "Afficher les statistiques"?

R4 : Comment "Choisir la destination la plus adaptée dans le cache"?

R4 : Comment "Choisir la destination la plus adaptée dans la table de routage"?

R4: Comment "Traiter l'argument reçu"?
Selon Tableau_commandes(Numero_argument) **Dans**
"-c" => (Taille_cache <- Integer'Value(Tableau_commandes(Numero_argument+1))
"-p" => (Politique <- Tableau_commandes(Numero_argument+1))
"-s" => (Statistique <- True)
"-S" => (Statistique <- False)
"-t" => (Table <- Tableau_commandes(Numero_argument+1))
"-p" => (Paquets <- Tableau_commandes(Numero_argument+1))
"-r" => (Resultats <- Tableau_commandes(Numero_argument+1))

R5 : Comment "Implanter dans le cache le dernier paquet utilisé"

R5 : Comment

Routeur LA avec la politique LRU:

R0: Utiliser un routeur LA en politique LRU

R1: Comment Utiliser un routeur LA en politique LRU?

Initialiser un routeur LA en politique LRU (Taille_cache : Integer, afficher_statistiques: Boolean, Table : Text_file, Paquets : Text_file, Résultats : Text_file, TableR : LA, SD_Cache : LA): **out**

Traiter les paquets Paquets : **in out**

Restituer les résultats Resultats : **in out**

R2: Comment "Initialiser un routeur LA en politique LRU?

Récupérer les caractéristiques, et les données (Taille_cache : Integer, Statistique : Boolean, Table : Text_file, Paquets : Text_file, Résultats : Text_file): **out**

Traiter l'ensemble des commandes reçues (Taille_cache : Integer, Statistique : Boolean, Table : Text_file, Paquets : Text_file, Résultats : Text_file): **in**

Initialiser la table de routage sous forme de liste chaînée (TableR) : **out**

Initialiser la structure de données qui représente le cache (SD_cache) : **out**

R2 : Comment "Traiter les paquets"?

Pour i **dans** 1..Nombre_paquets **Faire** - - On suppose que l'on connait le nombre total de paquets dans le fichier Paquets

Traiter le paquet_i Paquet_i : **in**

Mettre à jour le cache en fonction de la politique choisie

FinPour

R2: : Comment "Restituer les résultats"?
Pour i **dans** 1..Nombre_paquets **Faire**
Ecrire(Str(Paquet_i)) - - On affiche le paquet demandé et l'interface de destination sur la même ligne
Ecrire(Resultats(i)) - - Affiche l'interface de sortie
Retour à la ligne
Si Statistique = True **Faire**
Afficher les statistiques
Fin Si

R3 : Comment "Traiter le paquet_i"?
Choisir la destination la plus adaptée dans le cache (Route : Boolean, Destination_cache : String): **out,** (Paquet_i) : **in** - - Destination cache est l'interface d'arrivée
Si Route = False **faire**
Choisir la destination la plus adaptée dans la table de routage (Paquet_i :) **in**
Fin Si

R3: Comment "Récupérer les caractéristiques et les données" ?
Taille_cache <- 10 - - On met dans un premier temps toutes les valeurs par défauts
Statistique <- '-s'
Table <- table.txt
Paquets <- paquets.txt
Resultats <- resultats.txt

R3: Comment "Traiter l'ensemble des commandes reçues"?
Créer un tableau recensant l'ensemble des commandes (Tableau_commandes) : **out,**
Numero_argument <- 1
Tant que Numero_argument < Nombre_argument **Faire** - - Il y a maximum 5 arguments lors de l'appel de la fonction, on suppose Nombre_argument connu
Traiter l'argument reçu Numero_argument : **in**
Numero_argument <- Numero_argument +1
FinTantque

R3: Comment initialiser la table de routage sous forme de liste chaînée?
Initialiser (TableR) - - On initialise une liste chaînée vide (programme équivalent lors du mini projet 2)
Pour i dans 1..Nb_lignes_Table **Faire** - - On suppose que l'on connaît le nombre de lignes total du fichiers texte table
Enregistrer_Ligne_i
FinPour

R3: Comment "Initialiser la structure de données qui repéresente le cache"?
Initialiser(StructD_Cache)

R4: Comment "Afficher les statistiques"
Afficher le nombre de défauts de cache
Afficher le nombre de demandes de route
Afficher le taux de défaut de cache

R4 : Comment "Choisir la destination la plus adaptée dans la table de routage"?
Liste chainé

R4: Comment "Traiter l'argument reçu"?
Selon Tableau_commandes(Numero_argument) **Dans**
"-c" => (Taille_cache <-
Integer'Value(Tableau_commandes(Numero_argument+1))
"-s" => (Statistique <- True)
"-S" => (Statistique <- False)
"-t" => (Table <- Tableau_commandes(Numero_argument+1))
"-p" => (Paquets <- Tableau_commandes(Numero_argument+1))
"-r" => (Resultats <- Tableau_commandes(Numero_argument+1))

R4 : Comment "Implanter dans le cache le dernier paquet utilisé"

R4 : Comment "Choisir la destination la plus adaptée dans le cache"?
arbre

if cache_est_vide (in out booléen) then
- Initialiser le cache
- Trouver la destination
- Afficher la destination
- Gérer le cache selon une politique LRU

else
- Trouver la destination
- Afficher la destination
- Gérer le cache selon une politique LRU

R5 : Comment "Initialiser le cache"?
Initialiser le cache (racine)

R5 : Comment "Trouver la destination"?
se placer à la racine
while arbre(droite) != Null and arbre(gauche) != Null loop
if destination>arbre(centre) and arbre(droite) == Null then
comparer l'adresse IP avec la destination
elsif destination<arbre(centre) and arbre(gauche) == Null then
comparer l'adresse IP avec la destination
elsif destination>arbre(centre) then
aller_a_droite(arbre)

```
            elsif destination<arbre(centre) then
                    aller_a_gauche(arbre)
    else (if destination == arbre(centre))
            comparer l'adresse IP avec la destination
    end if
end loop
```
R5 : Comment "Afficher la destination"?
Afficher la destination

R5 : Comment "Gérer le cache selon une politique LRU"?

```
if taille_cache=taille_cache_init then
    supprimer la donnée la moins récemment utilisé
    ajouter 1 à la 'récentitude' de toutes les données
end if
if destination == arbre(centre) then
réinitialiser sa 'récentitude' à 0
else
ajouter la donnée à la detination avec un 'récentitude' de 0
end if
```

R6 : Comment "supprimer la donnée la moins récemment utilisée"?

```
ind_max = 0
for i in (1 .. Taille_cache) do
    if recentitude(arbre(i))>recentitude(arbre(ind_max)) then
            ind_max = i
    end if
end loop
supprimer(arbre(ind_max))
```

Commentaires sur les raffinages

Ces raffinages constituent la base du code général qui a suivi dans notre projet.

Architecture générale du code final

Le code général du projet est rédigé avec le language de programmation Ada. Dans un premier temps nous avons codé un routeur simple. Les différents modules créés ont servi au codage du routeur avec cache. La plupart des modules sont réutilisés dans le routeur avec cache.

Le routeur simple

Module table:

Dans ce module, nous définissons la liste chainée qui définira la table de routage. Il est composé des fonctions et procédures : Initialiser, Taille, Enregistrer, Supprimer, Vider, Route_Presente, LE_Masque, L_Interface, Chercher_Route.

Initialiser : est une procédure qui initialise une table vide.

Taille : est une fonction qui renvoie la taille de la table de routage.

Supprimer : est une procure prenant en argument la table que l'on souhaite modifier et la route que l'on souhaite supprimer de la table de routage.

Vider est une procédure qui supprime l'ensemble des routes de la table.

Route_Presente est une fonction qui renvoie un booléen indiquant si une route est pésente dans la table ou non.

Le_Masque est une fonction qui prend en paramètre une destination et retourne le masque associé à cette destination.

L_Interface est une fonction qui prend en paramètre une destination et qui retourne l'interface de sortie associée à cette destination.

Chercher_Route est une fonction qui renvoie l'interface de la route qui correspond à l'adresse ip du paquet envoyée selon les données des routes de la table de routage.

Module adresse ip:

Dans ce module, nous définissons le type adresse ip et l'ensemble des fonctions et procédures associée. Il y a aussi les fonctions et procédures liée à l'enregistrement et l'écrites des routes depuis ou vers un fichier texte.

Le type adresse ip est codé sur 32 bits. On regroupe ces 32 bits en 4 octets.

Le module adresse ip est composé des fonctions et procédures : Initialiser, Lire_adresse, Convertir_adresse, Compatible et « >=».

Initialiser est une procédure qui permet d'initialiser une adresse à partir des valeurs entières de chaque octet.

Lire_adresse est une procédure qui permet de lire une adresse dans un fichier texte et de l'initialiser en type adresse ip.

Convertir_adresse est une fonction qui permet de convertir une adresse de type adresse ip vers une chaine de caractère avant de l'insérer dans le fichier des résultats.

Compatible est une fonction qui permet de déterminer si une adresse avec un masque est compatible avec une destination.

«>= » est une fonction permettant de comparer deux données du type T_Adresse_IP et sera utilisée afin de comparer la taille de deux masques.

Le routeur avec un cache sous forme de liste chaînée

Dans cette partie du projet, nous avons codé le cache sous forme de liste chaînes. Nous avons donc du créer de nouveaux modules afin d'implanter l'utilisation du cache dans un programme nommé « routeur_l ».

Module Cache_L :

La procédure Initialiser permet d'initialiser un cache de taille 1 avec une cellule ayant pour valeur 0.

La fonction Cache_Plein renvoie un booléen signifiant si le cache est plein (et donc que pour ajouter une donnée il faut en supprimer une). La taille du cache est une valeur fixée par l'utilisateur (de base égale à 10) qui ne bouge pas une fois que la répartition des paquets commence.

La procédure Supprimer permet de supprimer une route précise du cache, En cas d'absence de cette route l'exception Route_Absente_Error est levée.

La procédure Supprimer_Derniere_Route supprime la ''dernière'' route du cache c'est à dire la plus ancienne. Cette procédure est utile dans l'application de la politique FIFO.

Les procédures Ajouter_Route_Debut et Ajouter_Route_Fin servent à ajouter une route au début ou à la fin du cache. Ces procédures seront

réutilisées pour enregistrer une route dans le cache ou pour mettre le cache à jour.

La procédure Augmenter_Frequence sert à augmenter la fréquence d'une route du cache de 1. Cette procédure sert à appliquer la politique LFU,

La procédure Supprimer_Minimum_Frequence sert à supprimer la route du cache avec la fréquence la moins élevée. Cette procédure sert à appliquer la politique LFU.

La fonction chercher sert à renvoyer la route du cache correspondant à une adresse saisie par l'utilisateur (si elle existe dans le cache ce qui n'est pas obligatoire).

La procédure Mettre_a_Jour sert à mettre à jour le Cache en fonction de sa politique. La Route prise en entrée est la route du cache venant d'être utilisée. Pour cette procédure on suppose que le cache est plein. Pour la politique LRU la procédure place la route au début du cache (en la supprimant puis en utilisant la procédure Ajouter_Route_Debut qui replace cette route au début du cache). Pour la politique LFU on augmente la fréquence de la route (procédure Augmenter_Frequence) . Pour la politique FIFO on supprime la dernière Route du cache (procédure Supprimer_Derniere_Route).

La procédure Enregistrer permet d'ajouter une route au cache. Pour la politiques FIFO on ajoute la route au début du cache, pour la politique LFU on supprime la route avec la fréquence la plus basse et on ajoute la route et pour la politique LFU on ajoute la route à la fin du cache.

La procédure Afficher_Cache permet d'afficher le cache à l'utilisateur.

La procédure Vider permet de supprimer toutes les routes du cache.

Le type T_Cache_L est un pointeur sur un type T_Cellule qui contient une donnée Route de type T_Route (définit dans le module complements_cache_l), une donnée Frequence entière et d'une donnée Suivant qui est de type T_Cache_L.

Module Complement_Cache_L :

Ce module contient les exceptions liées au cache qui sont : Option_Error lorsque l'utilisateur utilise mal les options, Route_Absente_Error lorsque le programme essayer de supprimer une route qui n'est pas présente dans le cache et Cache_Vide_Error lorsque le programme recherche une route dans un cache vide.

Ce module contient aussi la définition du type T_Route qui comprend les données Adresse et Masque de type T_Adresse_IP et la donnée Interface_Associee de type Unbounded_String.

Programme routeur.adb

Ce programme représente le routeur avec un cache de type listes chaînées.

La procédure Initialiser_Options permet d'initialiser les options ce qui permet à l'utilisateur de, par exemple, changer le nom du fichier de résultat, changer la politique du cache …

La procédure Importer_Table permet de mettre les données de la table au bon format (Adresse et Masque passent de Unbounded_String à T_Adresse_IP) et de les répartir afin de pouvoir les utiliser par la suite.

La procédure Enregistrer_Resultat permet d'enregistrer l'adresse de destination et l'interface associée à cette adresse dans le fichier de résultats.

Après avoir initialiser les paramètres à des valeurs par défaut on Initialise les options grâce à la procédure Initialiser_Options puis on importe la table (procédure Importer_Table) puis on associe chaque paquet avec une interface.

Une fois ces préparatifs achever on démarre une boucle tant que (qui s’achève à la fin du fichier contenant les paquets à router). Au début de la

boucle on transforme le paquet de Unbounded_String à T_Adresse_IP puis on cherche la Route correspondante dans le cache. Si l'interface associée à cette route est nulle on cherche alors la route dans la table de routage et on ajoute cette route au cache. Si l'interface n'est pas nulle alors la route correspondante se trouve dans le cache et on met à jour le cache.

Lorsque on a trouvé la route associée on utilise la procédure Enregistrer_Resultat afin de mettre cette route et l'interface associée dans le fichier résultat.

Le routeur avec un cache sous forme d'arbre

Cache Arbre :

Dans ce module, nous définissons l'arbre préfixe qui nous servira de cache. Il est composé des fonctions et procédures : Initialiser, Taille, Enregistrer, Supprimer_LRU, Vider, Ajouter_IP.

Initialiser : est une procédure qui initialise un arbre vide.

Taille : est une fonction qui renvoie la taille de l'arbre.

Supprimer_LRU: est une procure prenant en argument l'arbre que l'on souhaite modifier et la route que l'on souhaite supprimer de l'arbre qui est la route la moins récemment utilisé.

Vider est une procédure qui supprime l'ensemble des routes de l'arbre.

Ajouter_IP : Ajoute une adresse IP à l'arbre

Enregistrer : Enregistre un nouvel élément dans l'arbre en le mettant à la bonne place

L'arbre est utilisé en tant que cache, c'est un moyen d'éviter de sur utiliser la table de routage et de gagner en efficacité. Pour rester efficace le cache ne doit être trop rempli c'est pour cela qu'on lui assigne une taille, si la taille maximale

du cache est atteinte on supprime une donnée selon la politique choisi. Pour rédiger le code j'ai tenté d'abords de n'utiliser qu'une politique la LRU, c'est à dire qu'on va supprimer la donnée la moins récemment utilisée. L'arbre permet de se déplacer pour chercher une donnée on modélise la droite de l'arbre comme un 1 et se gauche comme un 0 et à l'aide de l'adresse IP on se déplace soit à droite soit à gauche. Pour éviter de se déplacer sur 32 bits, quand on enregistre un information on s'arrête dès que le chemin est suffisant pour qu'il n'y ai pas d'ambiguïté entre deux valeurs.

Les difficultés résidaient surtout pour la définition du type Arbre cumulé à la compréhension et à l'utilisation du type adresse IP. Logique des différentes fonctions naviguant dans l'arbre. D'ailleurs le code ne compile pas à cause des adresses IP entre autres, et malgré l'aide de mes camarades je n'ai pas pu saisir entièrement le principe.