

Rapport

Stockage et exploitation de tables de routage

réalisé par

Samy Afker, Maxime Moshfeghi et Ewan de Quillacq



Table des matières

Résumé	3
I Introduction	4
II Routeur	6
1 Fonctionnement du routeur	7
2 Modules implantés	8
2.1 Structure des adresses IP	8
2.2 Structure de listes chaînées associatives	8
2.3 Structure d'arbres binaires	9
3 Fichiers en entrée et en sortie	12
3.1 Table de routage	12
3.2 Paquets	12
3.3 Résultats	13
4 Fonctionnement du cache	14
4.1 Politiques de cache	14
4.2 Cohérence du cache	14
5 Programme principal	16
5.1 Traitement de la ligne de commande	16
5.2 Traitement du fichier contenant les paquets	17
III Utilisation du routeur	18
1 Installation	20
1.1 Compilation et première exécution	20
1.2 Test des instructions basiques	22
2 Test du routeur	25
2.1 Routeur simple	25
2.2 Politique FIFO	26
2.2.1 Exécution	26
2.2.2 Découpe du processus	27
2.3 Politique LRU	30

2.3.1	Exécution	30
2.3.2	Découpe du processus	30
2.4	Politique LFU	33
2.4.1	Exécution	33
2.4.2	Découpe du processus	35
2.5	Bilan des tests	38
IV	Conclusion	40
1	Nos programmes	41
2	Difficultés rencontrées	41
3	Nos expériences personnelles	41
3.1	Samy	41
3.2	Maxime	41
3.3	Ewan	41
4	Remerciements	42
A	Raffinages	43
B	Architecture du routeur	56

Résumé

Ce projet est une mise en pratique de l'ensemble des connaissances acquises en programmation impérative, notamment, les structures de données dynamiques (liste chaînées et arbre binaires). Le but ici sera d'implanter un programme pouvant router des adresses IP vers la bonne interface de sortie. Deux versions du routeur seront implantées, une première où le cache du routeur est sous forme d'arbre binaire et une autre où il a la forme d'une liste chaînée.

Première partie

Introduction

Le but de ce projet (réalisé en trinôme) est de coder en **Ada** un programme qui permet de router des adresse IP en utilisant une table de routage et en s'appuyant sur un cache.

La table de routage sera représentée par une liste chaînée et le cache sera disponible en deux version : liste chaînée ou arbre binaire.

Nous allons réaliser deux versions du routeur. Dans un premier temps, nous allons réaliser une première version simple du routeur ne disposant pas du cache. Ensuite, une version avec cache sera implantée.

Deuxième partie

Routeur

Chapitre 1

Fonctionnement du routeur

Un routeur est un élément d'un réseau qui a pour objectif de transmettre les paquets qu'il reçoit sur une interface d'entrée vers la bonne interface de sortie en fonction des informations qui sont stockées dans sa [table de routage](#). Pour des raisons de simplification, nous allons considérer que 3 informations essentielles : l'adresse, le masque et l'interface.

Routeur simple :

Dans un premier temps, nous avons réalisé une version simple du routeur. Cette dernière cherche la route correspondante à une adresse uniquement dans la table de routage puis écrit l'interface correspondante dans le fichier des résultats.

Routeur avec cache :

Pour améliorer l'efficacité du routeur, on utilise un cache. Le cache conserve un sous-ensemble des informations de la table de routage, celles qui ont des chances d'être utilisées dans le futur. Généralement, on conserve les dernières informations utilisées. L'idée est que si une information a été utilisée, elle a de bonnes chances d'être utilisée de nouveau.

Chapitre 2

Modules implantés

2.1 Structure des adresses IP

Il est dans un premier temps nécessaire de définir correctement les adresses manipulées. L'idée est de manipuler des adresses structurées de la manière suivante : `XXX.XXX.XXX.XXX`, où chaque `XXX` est un entier compris entre 0 et 255. Le module `adresse_ip`, spécifié dans le fichier `adresse_ip.ads` introduit donc trois sous-programmes principaux :

- la procédure `Afficher_IP` prenant en entrée une adresse IP et permettant de l'afficher à l'écran
- la fonction `Creer_Adresse` qui d'une chaîne de caractère en entrée de type `Unbounded_String` bien formulée renvoie l'objet de type `T_Adresse_IP` associé
- la fonction `Creer_Masque` qui d'une adresse IP en entrée (de type `T_Adresse_IP`) renvoie le masque correct associé.

Ces sous-programmes sont ensuite implémentés dans le fichier `adresse_ip.adb`.

2.2 Structure de listes chaînées associatives

Pour réaliser notre routeur, nous avons implanté dans un premier temps un module `p_routeur_11` dans lequel nous avons définis l'ensemble des objets utilisés. Le cache et la table de routage auront le type `T_LCA`, des listes chaînées associatives pointant vers un enregistrement qui contient :

- une adresse IP de type `T_Adresse_IP`
- le masque associé de type `T_Adresse_IP`

- l'interface vers laquelle l'adresse doit être dirigée de type `Unbounded_String`
- la fréquence d'utilisation de l'adresse de type `Integer` (usage exclusif à certains types de cache)
- la date de dernière utilisation de l'adresse de type `Integer` (usage également exclusif à certains types de cache)

Structure de l'enregistrement pointé par le type `T_LCA`

Adresse	Masque	Interface	Fréquence	Date	Suivant
---------	--------	-----------	-----------	------	---------

Le module comporte également un ensemble de procédures et de fonctions permettant entre autres d'agir sur les LCA, c'est-à-dire les initialiser, enregistrer une adresse et ses données associées, supprimer une certaine adresse, etc...

La spécification du module est donnée dans le fichier `p_routeur_11.ads`, et l'implémentation de l'ensemble des fonctions associées dans le fichier `p_routeur_11.adb`.

On note que ce second module fait appelle au module `adresse_ip` présenté précédemment.

2.3 Structure d'arbres binaires

Le type arbre binaire est un type récursif permettant d'accéder à 0, 1 ou 2 élément de ce même type. Il est composé de deux élément :

- les **nœuds**
- les **feuilles** (on les considéra comme vide ici)

Les nœuds amènent à 0, 1 ou 2 autres nœuds et peuvent contenir n'importe quelle information. Voici un exemple arbre binaire où les nœuds contiennent des entiers :

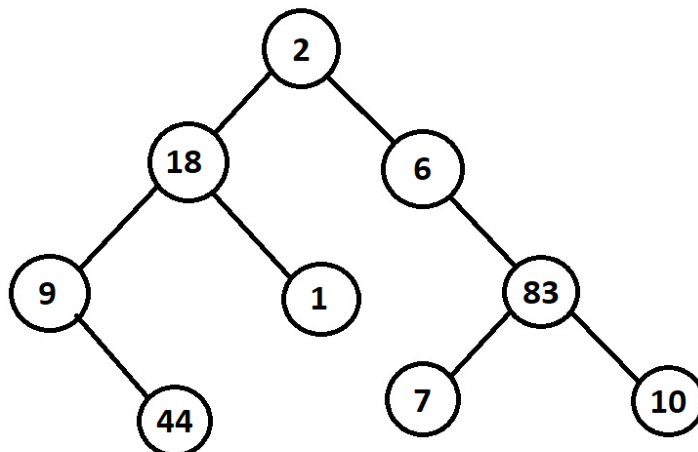


FIGURE 2.1 – Exemple d'arbre binaire

L'arbre que nous utilisons est un arbre binaire spécifique. Les informations dans les nœuds contiennent en particulier des adresse IP.

Le chemin prit pour accéder à ces adresses en question dépendent de leurs codage binaire (si le premier bit de l'adresse est un 1 on va à droite si ensuite c'est un 0 on va à gauche etc). On descend dans l'arbre jusqu'à ce que le préfixe de ce codage soit unique (aucune autre adresse possède le même préfixe). Ainsi seul les nœuds les plus profonds possèdent une information. Voici un exemple :

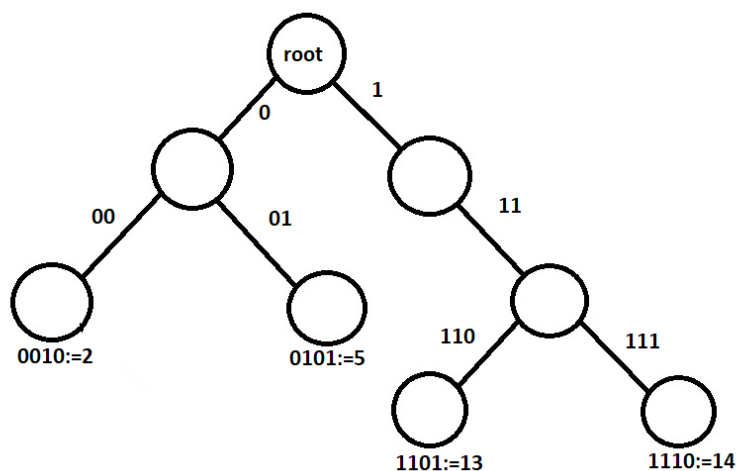


FIGURE 2.2 – Arbre binaire suivant le processus décrit ci-avant

Pour implanter ce type d'arbre (**T_arbre**) nous avons décidé de le définir comme un pointeur vers une cellule (type **T_cellule**). Cette cellule est un enregistrement qui contient plusieurs information comme :

- l'**adresse** (type **T_adresse_IP**)
- le **masque** (type **T_adresse_IP**)
- la **fréquence d'utilisation** dans le cache (type **Integer**)
- son **interface** (type **Unbounded_String**)
- la **date de dernière utilisation** de type **Integer**
- les arbres fils gauche et droit **SuivantG** et **SuivantD** de type **T_arbre**

On ne pourra qu'accéder à l'information du nœud où l'on se trouve (appelé racine ou root de l'arbre) ainsi pour accéder a toutes les informations il faudra parcourir tous les espaces pointés par le nœud et ainsi de suite.

Structure de l'enregistrement pointé par le type **T_arbre**

Adresse	Masque	Interface	Fréquence	Date	SuivantG	SuivantD
---------	--------	-----------	-----------	------	----------	----------

Chapitre 3

Fichiers en entrée et en sortie

3.1 Table de routage

Les tables de routages données en entrée du routeur seront des fichiers texte du format suivant :

Colonne 1	Colonne 2	Colonne 3
Destination	Masque	Interface
Destination	Masque	Interface
...

On note que deux colonnes sont séparées par un ou plusieurs espaces, et que chaque ligne peut également être précédée ou non d'un ou plusieurs espaces (cela ne gêne pas le fonctionnement du programme).

Exemple d'une ligne de la table de routage

<u>132.223.47.0</u>	<u>255.255.255.0</u>	<u>eth1</u>
Destination	Masque associé	Interface

Les trois premiers octets de l'adresse IP de destination étant non nuls, les trois premiers octets du cache associé sont paramétrés à 255.

3.2 Paquets

Le fichier contenant les paquets donné en entrée quant à lui ne sera composé que d'une seule colonne. Chaque ligne peut contenir une adresse ou une instruction parmi les suivantes :

- `table` : permet d'afficher la table de routage ;

- cache : permet d’afficher le cache ;
- stat : permet d’afficher les statistiques relatives au cache ;
- fin : indique la fin du programme ;

Destination 1
Instruction
Destination 3
Destination 3
Instruction
Destination 3
...

3.3 Résultats

Enfin, le résultat du routage est enregistré dans un fichier texte contenant deux colonnes, la première contenant l’adresse de destination des paquets donnés en entrée, et dans la deuxième l’interface qui lui est associée.

Colonne 1	Colonne 2
Destination 1	Interface 1
Destination 2	Interface 2
...	...

Remarques

- Si un paquet présente une adresse de destination incompatible avec les règles de routage, il n’apparaît pas dans le fichier résultat. En revanche, le terminal annoncera qu’un paquet n’a pas pu être routé.
- De même, les instructions inscrites dans le fichier contenant les paquets servant à afficher la table de routage, le cache ou encore les statistiques n’agissent d’aucune manière sur le fichier résultat.
- Seul l’instruction `fin` agit de sorte à ce que les paquets et instructions donnés en aval de l’instruction ne soient pas traités, et arrête alors le programme.

Chapitre 4

Fonctionnement du cache

4.1 Politiques de cache

Le cache dispose d'une capacité limitée, il sera donc nécessaire à un moment donné de supprimer une adresse qui y est présente afin de libérer de la place pour la prochaine adresse à enregistrer. Pour cela, plusieurs politiques peuvent être adoptées :

- **FIFO** : First In First Out, ce qui signifie que le routeur supprime la première adresse insérée dans le cache.
- **LFU** : Least Frequently Used, ici le routeur supprime l'adresse utilisée le moins fréquemment.
- **LRU** : Last Recently Used, ici le routeur supprime l'adresse utilisée le moins récemment.

4.2 Cohérence du cache

Une propriété importante du cache consiste à savoir si une donnée qui s'y trouve est valide ou non. On parle de cohérence du cache. Pour garantir la validité des données du cache, il suffit de s'assurer que les données insérées sont correctes. Prenons pour exemple la table de routage suivante :

Destination	Masque	Interface
147.127.0.0	255.255.0.0	eth1
147.127.18.0	255.255.255.0	eth2

On suppose le cache initialement vide et une demande de route pour la destination 147.127.25.12 . Seule la première route correspond. On pourrait donc ajouter cette route dans le cache. Supposons maintenant que le routeur reçoive une demande de route pour la destination

147.127.18.85 . La recherche dans le cache fournit une seule route qui conduit à utiliser l'interface `eth1`. Cependant, d'après la table de routage complète, l'interface à prendre est `eth2`. En effet, les deux routes correspondent et c'est celle du masque le plus long qu'il faut utiliser. Dans cet exemple, la route qu'il aurait fallu mettre en cache était :

147.127.25.0 255.255.255.0 `eth1`

Afin de résoudre ce problème, nous avons défini une précision pour le cache. Cette dernière est calculée une seule fois au début du programme et correspond au nombre minimal de zéro à la fin des adresses de la table de routage. Prenons pour exemple la table de routage suivante :

Destination	Masque	Interface
147.127.0.0	255.255.0.0	<code>eth1</code>
147.127.18.0	255.255.255.0	<code>eth2</code>
192.0.0.0	255.0.0.0	<code>eth3</code>
0.0.0.0	0.0.0.0	<code>eth0</code>

Ici la précision est 1. En effet, c'est la deuxième adresse de la table de routage qui contient le nombre minimale de zéro.

Après avoir calculé la précision du cache, la route à ajouter dans le cache sera complétée à cette précision près. Par exemple, considérons l'adresse 192.168.1.10 , la route à utiliser depuis la table de routage est donc 192.0.0.0 . Finalement, la route qu'on devrait ajouter dans le cache est donc 192.168.1.0 .

Chapitre 5

Programme principal

5.1 Traitement de la ligne de commande

Dans un premier temps, l'idée est de traiter l'ensemble des requêtes données par l'utilisateur qui peut entre autres :

- changer les noms des fichiers en entrée et en sortie avec les instruction `-p|-t|-r <nom du fichier>`, paramétrés par défaut pour les paquets, table de routage et résultats du routage respectivement sur `paquets.txt`, `table.txt` et `resulats.txt`
- changer la taille du cache en donnant l'instruction `-c <taille>` où `<taille>` est un entier qui vaut 10 par défaut
- changer la politique de cache utilisée en précisant `-P <FIFO|LRU|LFU>`
- afficher ou non les statistiques par les instructions respectives `-s` ou `-S` ; par défaut, le programme affichera les statistiques
- afficher l'aide associée au programme avec l'instruction `-help`

La ligne de commande est traitée par une structure de contrôle Pour qui itère sur chaque argument de la ligne de commande (chaque chaîne de caractères délimitée par un espace est considéré comme un argument). Dans le cas où l'on donne deux instructions relatives à un même paramètre, c'est la dernière qui est enregistrée.

Exemple

Si l'on rentre :

$$\underbrace{./routeur_ll}_{\text{exécution}} \quad \underbrace{-c \ 10}_{(1)} \quad \underbrace{-c \ 25}_{(2)}$$

où l'on demande successivement deux instructions différentes pour la taille de cache (instructions **(1)** et **(2)**), la taille retenue par le programme pour le cache sera la dernière, à savoir ici 25. Les autres paramètres seront assignées à leur valeur par défaut (le programme donnera ses résultats dans le fichier `resultats.txt`, etc...)

Les valeurs ensuite enregistrées pour chacun des paramètres sont ensuite traitées, c'est-à-dire que successivement :

1. La table de routage est créée (conversion du fichier contenant la table de routage en une table exploitable par notre programme de type `T_LCA`).
2. Ensuite, le fichier qui va contenir les résultats est créé, et le fichier contenant les paquets est ouvert.
3. Enfin, le routage des adresses contenues dans le fichier paquets commence.

5.2 Traitement du fichier contenant les paquets

Le fichier contenant les paquets est ensuite lu par itération sur chacune des lignes du fichier à l'aide d'une structure de contrôle **Tant Que**. On sort alors de la boucle lorsque le fichier est entièrement lu ou lorsque le programme rencontre une ligne contenant le mot `fin`.

Troisième partie

Utilisation du routeur

Remarque préliminaire

Pour les tests, la table de routage utilisée sera la suivante :

Listing 1 – Table de routage utilisée pour l'ensemble des test

1	132.223.47.0	255.255.255.0	eth1
2	141.127.16.0	255.255.240.0	eth2
3	149.127.18.0	255.255.255.0	eth3
4	149.127.0.0	255.255.0.0	eth4
5	222.123.0.0	255.255.0.0	eth5
6	0.0.0.0	0.0.0.0	eth0

Chapitre 1

Installation

1.1 Compilation et première exécution

Initialement, le dossier contient l'ensemble des fichiers suivant :

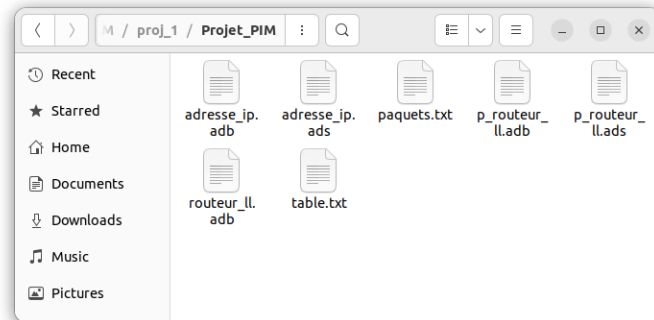
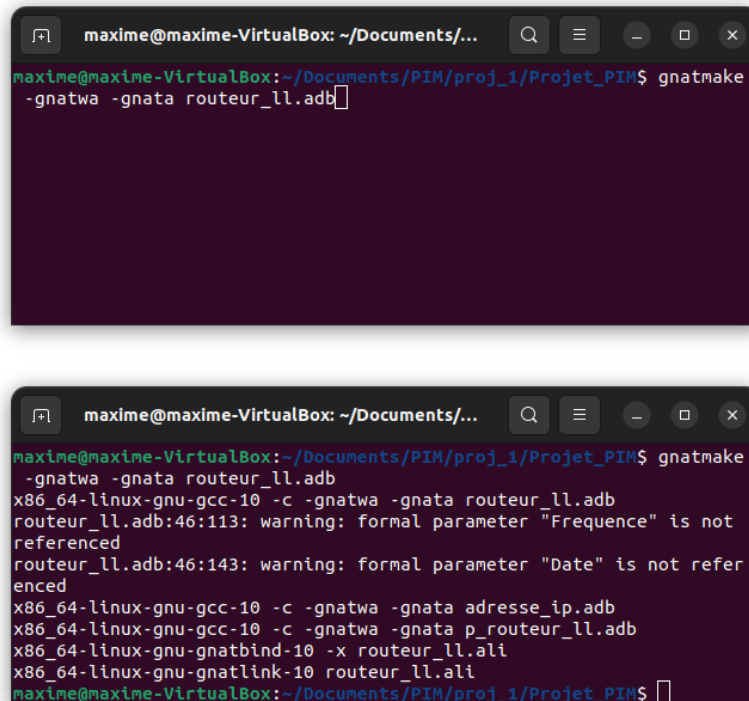


FIGURE 1.1 – Contenu initial du dossier courant

On note qu'il contient en particulier l'ensemble des modules (spécification et implémentation), le programme que l'on a nommé `routeur_ll` ainsi que deux fichiers qui nous serviront de tests créés à la main `table.txt` et `paquets.txt`.

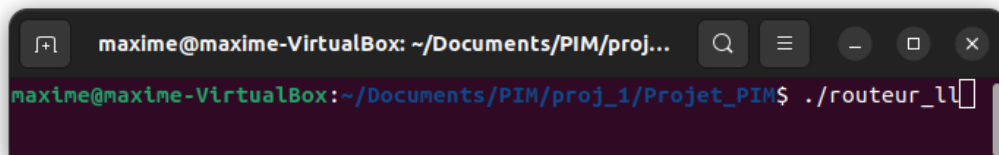
On compile ensuite le programme :



```
maxime@maxime-VirtualBox: ~/Documents/...  
maxime@maxime-VirtualBox:~/Documents/PIM/proj_1/Projet_PIM$ gmake  
-gnatwa -gnata routeur_ll.adb  
  
maxime@maxime-VirtualBox:~/Documents/PIM/proj_1/Projet_PIM$ gmake  
-gnatwa -gnata routeur_ll.adb  
x86_64-linux-gnu-gcc-10 -c -gnatwa -gnata routeur_ll.adb  
routeur_ll.adb:46:113: warning: formal parameter "Frequence" is not  
referenced  
routeur_ll.adb:46:143: warning: formal parameter "Date" is not refer  
enced  
x86_64-linux-gnu-gcc-10 -c -gnatwa -gnata adresse_ip.adb  
x86_64-linux-gnu-gcc-10 -c -gnatwa -gnata p_routeur_ll.adb  
x86_64-linux-gnu-gnatbind-10 -x routeur_ll.ali  
x86_64-linux-gnu-gnatlink-10 routeur_ll.ali  
maxime@maxime-VirtualBox:~/Documents/PIM/proj_1/Projet_PIM$
```

FIGURE 1.2 – Compilation

Puis on l'exécute :



```
maxime@maxime-VirtualBox: ~/Documents/PIM/proj...  
maxime@maxime-VirtualBox:~/Documents/PIM/proj_1/Projet_PIM$ ./routeur_ll
```

FIGURE 1.3 – Exécution de routeur_ll

Après exécution on doit avoir l'ensemble de ces fichiers dans le répertoire courant :

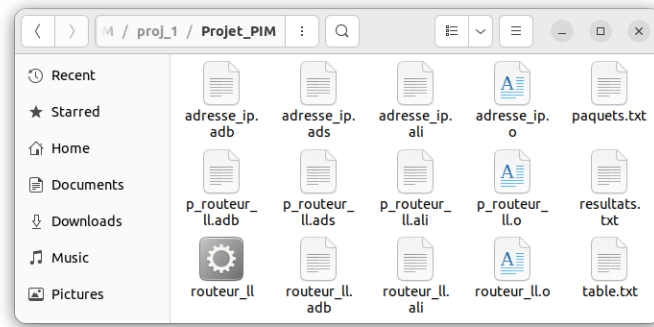


FIGURE 1.4 – Contenu final du dossier

En particulier, on remarque qu'un fichier **resultat.txt** a été créé (les autres fichiers sont issus de la compilation du programme).

Remarque

Si l'on souhaite compiler ou exécuter le routeur avec un cache de type arbre binaire, le programme aura comme nom **routeur_la.adb**.

1.2 Test des instructions basiques

On s'attarde dans un premier temps sur le bon fonctionnement des instructions simples que l'on peut mettre dans le fichier contenant les paquets. On demande donc d'exécuter le programme avec les paramètres suivants :

```
./routeur_ll -S -c 5 -t t_TEST.txt -p p_PREMIER_TEST.txt -r r_PREMIER_TEST.txt
```

exécution
(1)
(2)
(3)
(4)
(5)

Dans cet appel au programme :

- l'instruction **(1)** sert à préciser que l'on ne souhaite pas afficher les statistiques en fin d'exécution (elles pourront cependant l'être dans le cas d'une instruction **stat** dans un fichier paquets)
- l'instruction **(2)** sert à paramétrer la taille maximale du cache à 5
- les instructions **(3)**, **(4)** et **(5)** servent à indiquer les noms des fichiers en entrée et de celui en sortie.

Le fichier paquets lu par le programme dans ce premier test est le suivant :

Listing 1.1 – Paquets à router pour le test des instructions basiques

```

1 table
2 149.127.25.35
3 149.127.18.29
4 141.127.16.12
5 132.223.47.40
6 cache
7 fin
8 46.53.175.49
9 stat

```

On obtient alors en fichier résultat :

Listing 1.2 – Résultat du routage pour le test des instructions basiques

```

1 149.127.25.35      eth4
2 149.127.18.29      eth3
3 141.127.16.12      eth2
4 132.223.47.40      eth1

```

Dans un premier temps, on remarque le fichier résultats contient 4 adresses routées, ce qui permet de voir que l’instruction `fin` a bien été interprétée puisque le paquet présent en aval de celle-ci n’a pas été routé.

On s’intéresse maintenant à ce qui a été affiché dans le terminal. Dans un premier temps, il est demandé dans le fichier paquets d’afficher la table, ce qui fonctionne correctement :

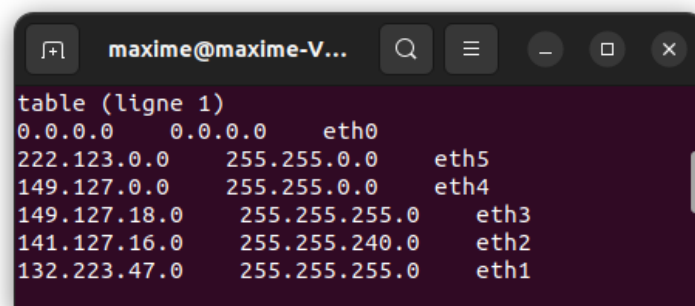
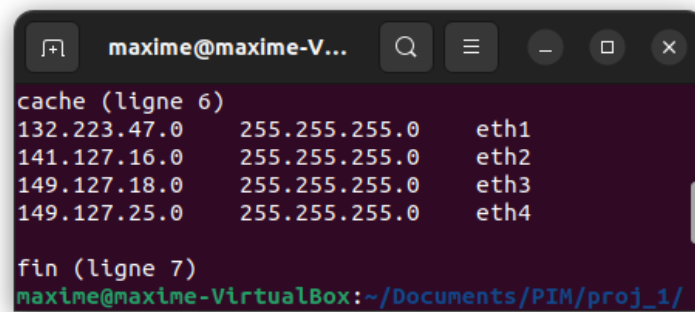


FIGURE 1.5 – Instruction `table`

On voit de même que le cache est affiché au moment de la demande :

A terminal window titled 'maxime@maxime-V...' with standard window controls. The terminal output shows a 'cache' instruction followed by a table of IP addresses and interface names. The table has three columns: IP address, another IP address (255.255.255.0), and an interface name (eth1, eth2, eth3, eth4). The output ends with a 'fin' instruction and the terminal prompt 'maxime@maxime-VirtualBox:~/Documents/PIM/proj_1/'.

```
maxime@maxime-V...  
cache (ligne 6)  
132.223.47.0      255.255.255.0    eth1  
141.127.16.0      255.255.255.0    eth2  
149.127.18.0      255.255.255.0    eth3  
149.127.25.0      255.255.255.0    eth4  
fin (ligne 7)  
maxime@maxime-VirtualBox:~/Documents/PIM/proj_1/
```

FIGURE 1.6 – Instruction `cache`

On voit aussi que l’instruction `fin` donné dans le paquet est notifié dans le terminal et signale au passage la ligne à partir de laquelle le routage s’est arrêté.

On aurait également pu montrer que l’instruction `stat` fonctionne correctement, mais nous aurons le temps de le constater grâce aux multiples test de cache qui vont suivre.

Chapitre 2

Test du routeur

2.1 Routeur simple

Pour exécuter le routeur simple, il faut demander une taille de cache de 0 :

$\underbrace{./\text{routeur_11}}_{\text{exécution}} \underbrace{-c\ 0}_{\text{simple}}$

On va essayer de demander au routeur simple d’afficher le cache et les statistiques pour vérifier que celui-ci ne s’arrête pas en lisant ces instructions. Voici donc le fichier paquets qu’on va lui donner :

Listing 2.1 – Paquets donnés au routeur simple

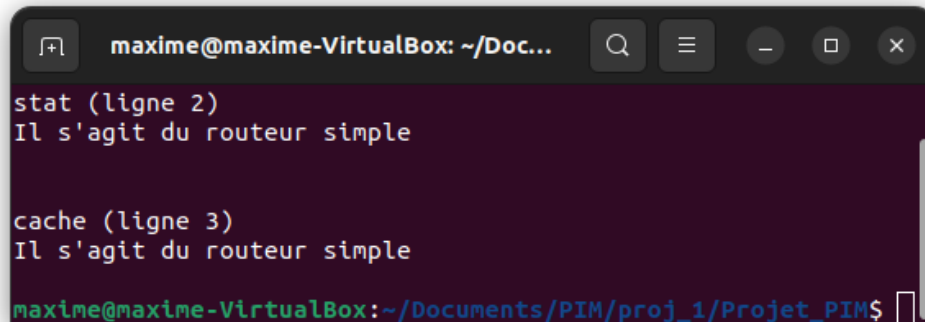
```
1 149.127.41.57
2 stat
3 cache
4 74.22.47.40
```

On constate effectivement que cela a fonctionné, puisqu’on obtient dans le fichiers résultat :

Listing 2.2 – Résultats obtenus avec le routeur simple

```
1 149.127.41.57      eth4
2 74.22.47.40       eth0
```

De plus, dans le terminal, on a été notifié que les deux instructions `stat` et `cache` n'ont pas pu être interprétées :



```
maxime@maxime-VirtualBox: ~/Doc...  
stat (ligne 2)  
Il s'agit du routeur simple  
  
cache (ligne 3)  
Il s'agit du routeur simple  
maxime@maxime-VirtualBox:~/Documents/PIM/proj_1/Projet_PIM$
```

FIGURE 2.1 – Exécution du routeur simple

2.2 Politique FIFO

2.2.1 Exécution

Dans un premier temps, nous allons tester le bon fonctionnement de la politique FIFO. Voici donc le fichier contenant les paquets édités que nous allons tester :

Listing 2.3 – Paquets à router pour le test de la politique FIFO

```
1 149.127.25.35  
2 149.127.18.29  
3 141.127.16.12  
4 132.223.47.40  
5 222.123.42.59  
6 stat  
7 cache  
8 132.223.47.45  
9 stat  
10 cache  
11 222.123.88.102  
12 stat  
13 cache  
14 222.123.50.233  
15 stat  
16 cache
```

```
17 147.127.18.32
18 stat
19 cache
```

On obtient alors le fichier contenant les résultats suivant :

Listing 2.4 – Résultat du routage avec la politique FIFO

```
1 149.127.25.35      eth4
2 149.127.18.29      eth3
3 141.127.16.12      eth2
4 132.223.47.40      eth1
5 222.123.42.59      eth5
6 132.223.47.45      eth1
7 222.123.88.102     eth5
8 222.123.50.233     eth5
9 147.127.18.32      eth0
```

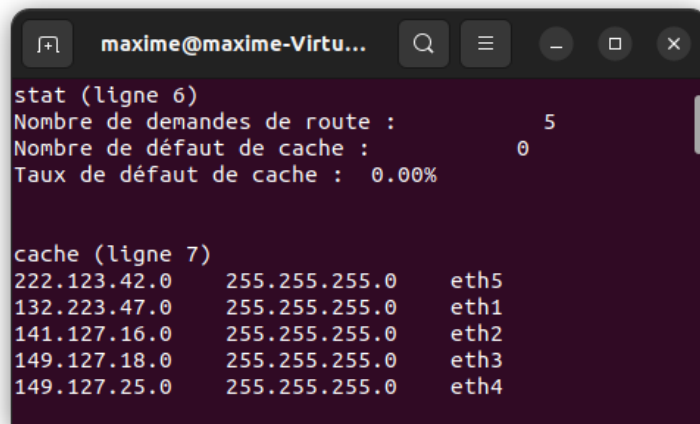
Après avoir vérifié que chaque destination était routée sur la bonne interface, nous allons découper l'exécution du routeur.

2.2.2 Découpe du processus

Pour ce faire, nous allons dans un premier temps remplir le cache, dont on a défini la taille maximale à 5 :

Listing 2.5 – Fichier paquets, ligne 1 à 7

```
1 149.127.25.35
2 149.127.18.29
3 141.127.16.12
4 132.223.47.40
5 222.123.42.59
6 stat
7 cache
```



```

maxime@maxime-Virtu...
stat (ligne 6)
Nombre de demandes de route :      5
Nombre de défaut de cache :        0
Taux de défaut de cache :  0.00%

cache (ligne 7)
222.123.42.0    255.255.255.0    eth5
132.223.47.0    255.255.255.0    eth1
141.127.16.0    255.255.255.0    eth2
149.127.18.0    255.255.255.0    eth3
149.127.25.0    255.255.255.0    eth4

```

FIGURE 2.2 – Remplissage du cache

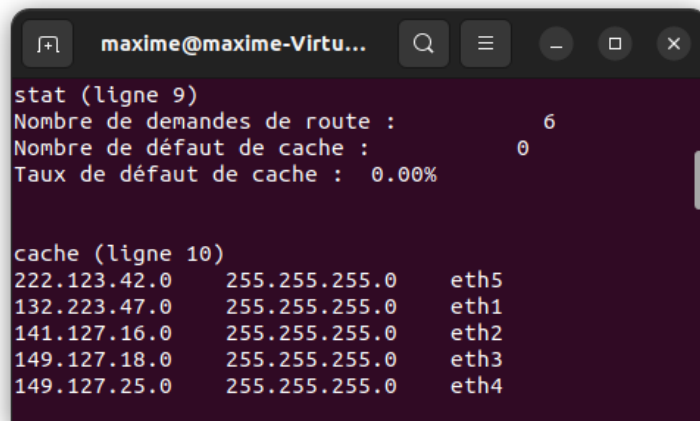
Nous pouvons alors vérifier que l'ajout d'un paquet dont la route est connue par le cache ne le modifie pas :

Listing 2.6 – Fichier paquets, ligne 8 à 10

```

8 132.223.47.45
9 stat
10 cache

```



```

maxime@maxime-Virtu...
stat (ligne 9)
Nombre de demandes de route :      6
Nombre de défaut de cache :        0
Taux de défaut de cache :  0.00%

cache (ligne 10)
222.123.42.0    255.255.255.0    eth5
132.223.47.0    255.255.255.0    eth1
141.127.16.0    255.255.255.0    eth2
149.127.18.0    255.255.255.0    eth3
149.127.25.0    255.255.255.0    eth4

```

FIGURE 2.3 – Route connue par le cache

On va donc ajouter une adresse que le cache ne connaît pas :

Listing 2.7 – Fichier paquets, ligne 11 à 13

```

11 222.123.88.102
12 stat

```

13 | cache

```

maxime@maxime-Virtu...
stat (ligne 12)
Nombre de demandes de route :      7
Nombre de défaut de cache :        1
Taux de défaut de cache : 14.29%

cache (ligne 13)
222.123.88.0    255.255.255.0    eth5
222.123.42.0    255.255.255.0    eth5
132.223.47.0    255.255.255.0    eth1
141.127.16.0    255.255.255.0    eth2
149.127.18.0    255.255.255.0    eth3

```

FIGURE 2.4 – Appel d’une adresse non connue par le cache

On remarque alors que le nombre de défaut de cache est passé de 0 à 1 : la première route a donc été supprimée du cache.

On peut ajouter d’autres destination non présente dans le cache, et on remarque alors l’évolution de celui-ci :

Listing 2.8 – Fichier paquets, ligne 14 à 19

```

14 222.123.50.233
15 stat
16 cache
17 147.127.18.32
18 stat
19 cache

```

```

maxime@maxime-Virtu...
stat (ligne 15)
Nombre de demandes de route :      8
Nombre de défaut de cache :        2
Taux de défaut de cache : 25.00%

cache (ligne 16)
222.123.50.0    255.255.255.0    eth5
222.123.88.0    255.255.255.0    eth5
222.123.42.0    255.255.255.0    eth5
132.223.47.0    255.255.255.0    eth1
141.127.16.0    255.255.255.0    eth2

```

```

maxime@maxime-Virtu...
stat (ligne 18)
Nombre de demandes de route :      9
Nombre de défaut de cache :        3
Taux de défaut de cache : 33.33%

cache (ligne 19)
147.127.18.0    255.255.255.0    eth0
222.123.50.0    255.255.255.0    eth5
222.123.88.0    255.255.255.0    eth5
222.123.42.0    255.255.255.0    eth5
132.223.47.0    255.255.255.0    eth1

```

FIGURE 2.5 – Appel d’autres adresses non connue par le cache

2.3 Politique LRU

2.3.1 Exécution

On va maintenant tester la politique LRU. On utilise cette fois-ci le fichier paquets suivant :

Listing 2.9 – Paquets à router pour le test de la politique LRU

```

1 141.127.16.12
2 149.127.18.29
3 149.127.25.35
4 132.223.47.40
5 222.123.42.59
6 stat
7 cache
8 141.127.16.45
9 stat
10 cache
11 100.100.100.100
12 stat
13 cache
14 fin

```

Et on obtient les résultats suivant :

Listing 2.10 – Routage des paquets de test LRU

```

1 141.127.16.12      eth2
2 149.127.18.29      eth3
3 149.127.25.35      eth4
4 132.223.47.40      eth1
5 222.123.42.59      eth5
6 141.127.16.45      eth2
7 100.100.100.100    eth0

```

On trouve que les adresses ont également été routées correctement.

2.3.2 Découpe du processus

Comme pour le test de la politique FIFO, on remplit dans un premier temps le cache :

Listing 2.11 – Fichier paquets, ligne 1 à 7

```

1 141.127.16.12
2 149.127.18.29
3 149.127.25.35
4 132.223.47.40
5 222.123.42.59
6 stat
7 cache

```

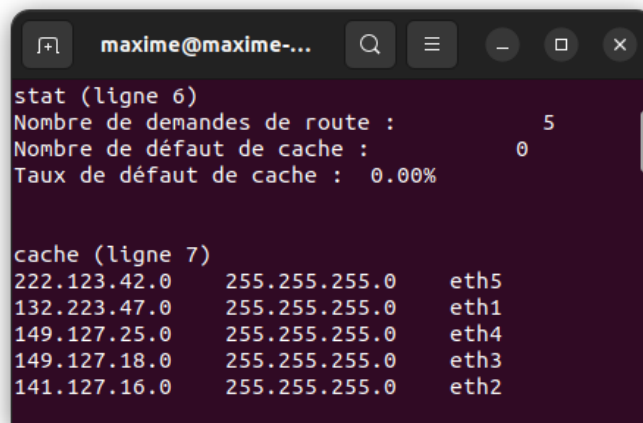


FIGURE 2.6 – Remplissage du cache

À ce stade de l'exécution, les routes enregistrées par le cache sont les suivantes, où l'ordre d'appel est le plus grand pour l'adresse ajoutée la plus récemment :

Destination	Masque	Interface	Ordre d'appel
141.127.16.0	255.255.255.0	eth2	1
149.127.18.0	255.255.255.0	eth3	2
149.127.25.0	255.255.255.0	eth4	3
132.223.47.0	255.255.255.0	eth1	4
222.123.42.0	255.255.255.0	eth5	5

Pour tester le bon fonctionnement de la politique LRU, on peut dans un premier temps appeler la première valeur enregistrée dans le cache, à savoir 141.127.16.45. L'idée est de réactualiser la date de dernier appel de cette adresse, qui passe donc de la moins récemment utilisée à la plus récemment utilisée.

Listing 2.12 – Fichier paquets, ligne 8 à 10

```

8 141.127.16.45
9 stat
10 cache

```

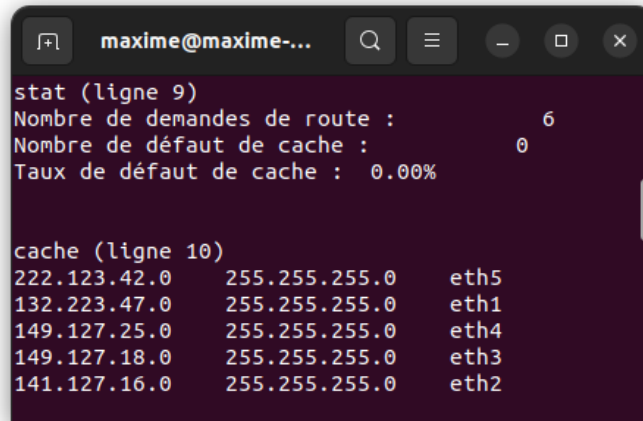


FIGURE 2.7 – Appel de l'adresse la moins récemment utilisée

La destination 141.127.16.45 est donc devenu l'adresse la plus récemment appelée. Les ordres sont donc devenus les suivants :

Destination	Masque	Interface	Ordre d'appel
141.127.16.0	255.255.255.0	eth2	6
149.127.18.0	255.255.255.0	eth3	1
149.127.25.0	255.255.255.0	eth4	2
132.223.47.0	255.255.255.0	eth1	3
222.123.42.0	255.255.255.0	eth5	4

On ajoute alors une adresse non connue par le cache. C'est alors l'adresse avec l'ordre d'appel le plus bas, à savoir ici 1 (dont la destination est 149.127.18.0) qui doit être supprimée du cache, et on le vérifie alors bien :

Listing 2.13 – Fichier paquets, ligne 11 à 13

```

11 100.100.100.100
12 stat
13 cache

```

```

maxime@maxime-...
stat (ligne 12)
Nombre de demandes de route :      7
Nombre de défaut de cache :        1
Taux de défaut de cache : 14.29%

cache (ligne 13)
100.100.100.0    255.255.255.0    eth0
222.123.42.0     255.255.255.0    eth5
132.223.47.0     255.255.255.0    eth1
149.127.25.0     255.255.255.0    eth4
141.127.16.0     255.255.255.0    eth2

```

FIGURE 2.8 – Appel d’une nouvelle adresse IP

2.4 Politique LFU

2.4.1 Exécution

On va enfin tester la politique LFU. On utilise alors le fichier paquets suivant :

Listing 2.14 – Paquets à router pour le test de la politique LFU

```

1 149.127.25.35
2 149.127.18.29
3 141.127.16.12
4 132.223.47.40
5 222.123.42.59
6 stat
7 cache
8 149.127.25.97
9 stat
10 cache
11 149.127.25.96
12 149.127.25.95
13 149.127.25.94
14 149.127.25.93
15 132.223.47.99
16 132.223.47.98
17 132.223.47.97
18 132.223.47.96

```

```

19 149.127.18.1
20 149.127.18.2
21 149.127.18.3
22 141.127.16.8
23 141.127.16.9
24 222.123.42.52
25 stat
26 cache
27 101.21.34.76
28 stat
29 cache
30 fin

```

Et on obtient les résultats suivant :

Listing 2.15 – Routage des paquets de test LFU

```

1 149.127.25.35      eth4
2 149.127.18.29      eth3
3 141.127.16.12      eth2
4 132.223.47.40      eth1
5 222.123.42.59      eth5
6 149.127.25.97      eth4
7 149.127.25.96      eth4
8 149.127.25.95      eth4
9 149.127.25.94      eth4
10 149.127.25.93      eth4
11 132.223.47.99      eth1
12 132.223.47.98      eth1
13 132.223.47.97      eth1
14 132.223.47.96      eth1
15 149.127.18.1       eth3
16 149.127.18.2       eth3
17 149.127.18.3       eth3
18 141.127.16.8       eth2
19 141.127.16.9       eth2
20 222.123.42.52      eth5
21 101.21.34.76       eth0

```

Là encore les adresses sont routées correctement.

2.4.2 Découpe du processus

Comme pour les deux politiques de cache précédentes, on remplit d'abord le cache :

Listing 2.16 – Fichier paquets, ligne 1 à 7

```

1 149.127.25.35
2 149.127.18.29
3 141.127.16.12
4 132.223.47.40
5 222.123.42.59
6 stat
7 cache

```

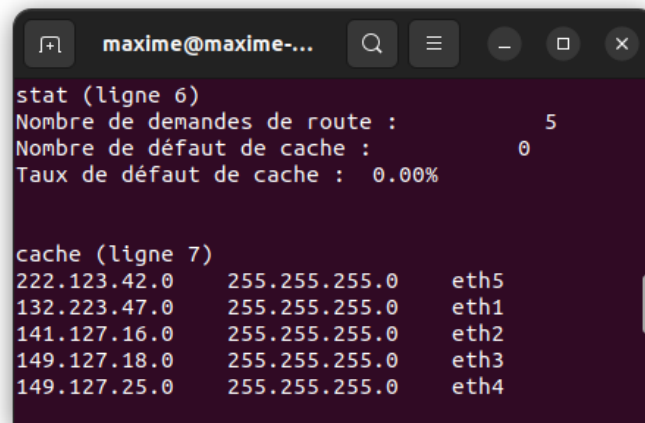


FIGURE 2.9 – Remplissage du cache

À ce stade de l'exécution, les fréquences d'appel de chacune des destinations sont égales à 1 :

Destination	Masque	Interface	Fréquence d'appel
149.127.25.0	255.255.255.0	eth4	1
149.127.18.0	255.255.255.0	eth3	1
141.127.16.0	255.255.255.0	eth2	1
132.223.47.0	255.255.255.0	eth1	1
222.123.42.0	255.255.255.0	eth5	1

Maintenant, l'idée est de vérifier que pour des routes enregistrées dans le cache aux fréquences d'appel différentes, à l'appel d'une adresse

IP non connue, c'est celle avec la plus petite fréquence qui est supprimée du cache.

Ainsi, l'appel suivant augmente de 1 la fréquence d'appel de la destination 149.127.25.97 :

Listing 2.17 – Fichier paquets, ligne 8 à 10

```
8 149.127.25.97
9 stat
10 cache
```

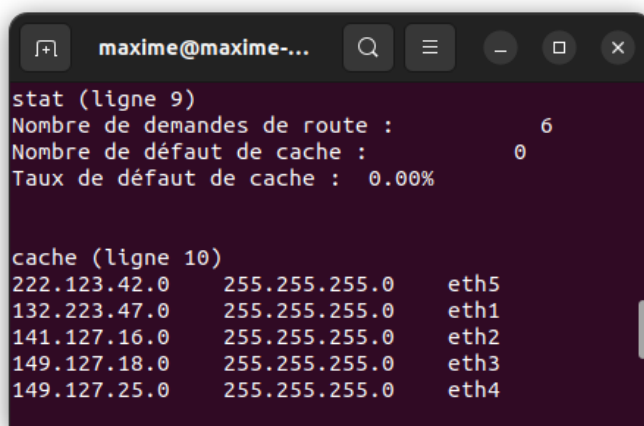


FIGURE 2.10 – Remplissage du cache

Destination	Masque	Interface	Fréquence d'appel
149.127.25.0	255.255.255.0	eth4	1
149.127.18.0	255.255.255.0	eth3	1
141.127.16.0	255.255.255.0	eth2	2
132.223.47.0	255.255.255.0	eth1	1
222.123.42.0	255.255.255.0	eth5	1

On appelle donc volontairement plusieurs fois chacune des adresses du cache :

Listing 2.18 – Fichier paquets, ligne 11 à 26

```
11 149.127.25.96
12 149.127.25.95
13 149.127.25.94
14 149.127.25.93
```

```

15 132.223.47.99
16 132.223.47.98
17 132.223.47.97
18 132.223.47.96
19 149.127.18.1
20 149.127.18.2
21 149.127.18.3
22 141.127.16.8
23 141.127.16.9
24 222.123.42.52
25 stat
26 cache

```

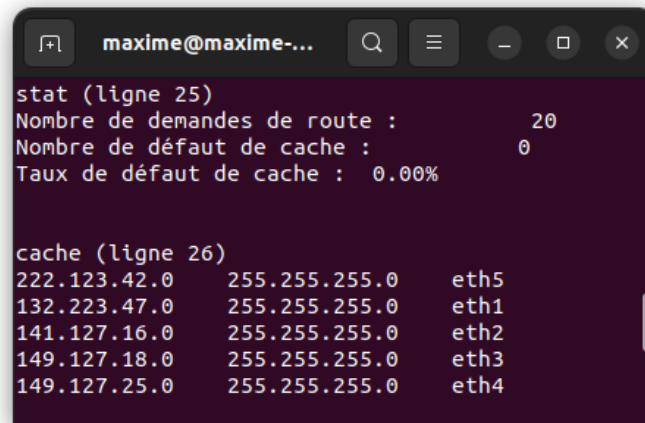


FIGURE 2.11 – Modification des fréquences des routes enregistrées dans le cache

Destination	Masque	Interface	Fréquence d'appel
149.127.25.0	255.255.255.0	eth4	6
149.127.18.0	255.255.255.0	eth3	4
141.127.16.0	255.255.255.0	eth2	3
132.223.47.0	255.255.255.0	eth1	5
222.123.42.0	255.255.255.0	eth5	2

Dans notre cache, la route la moins appelée est donc celle dont l'adresse est 222.123.42.0, et c'est donc celle qui doit être supprimée du cache si l'on demande une destination non connue par ce dernier. On vérifie donc cela aisément :

Listing 2.19 – Fichier paquets, ligne 27 à 29

```

27 101.21.34.76
28 stat
29 cache

```

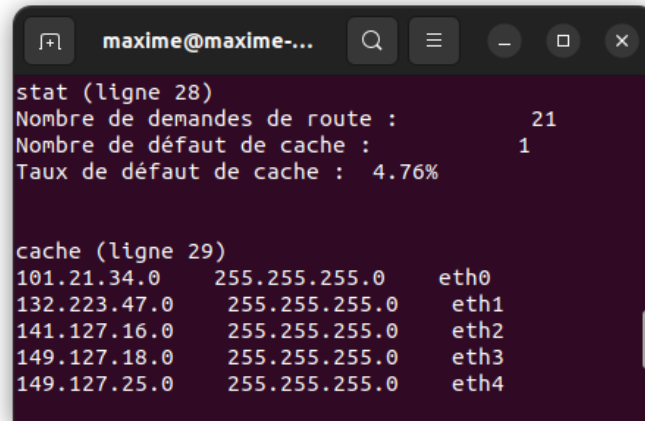


FIGURE 2.12 – Appel d’une nouvelle adresse IP

Destination	Masque	Interface	Fréquence d’appel
149.127.25.0	255.255.255.0	eth4	6
149.127.18.0	255.255.255.0	eth3	4
141.127.16.0	255.255.255.0	eth2	3
132.223.47.0	255.255.255.0	eth1	5
101.21.34.0	255.255.255.0	eth5	1

2.5 Bilan des tests

On a ainsi vérifié que les 3 politiques fonctionnent correctement. On peut décider d’utiliser une politique ou une autre selon l’usage que l’on compte faire du routeur. Chacun possède leurs qualités et leurs défauts. Par exemple, pour le cas de la politique LFU, on peut remarquer que si on utilise beaucoup certaines routes du cache, et que ces routes ne sont plus utilisées par la suite, si une nouvelle séquence de destinations sont demandées mais de manière moins fréquentes, le cache pourrait être rempli d’adresses qui ne sont plus utilisées depuis longtemps, pouvant d’ailleurs rendre les exécutions plus longues puisque le routeur sera contraint de parcourir la table de routage pour trouver la route convenable à chaque demande d’une nouvelle destination.

Il pourrait être possible de corriger ce genre de problème en combinant les idées des politiques LRU et LFU, en faisant par exemple décrémenter la donnée utilisée pour évaluer la fréquence d'appel au cours de l'exécution des paquets si celle-ci n'est pas rappelée dans un certain délai.

Quatrième partie

Conclusion

1 Nos programmes

Nous sommes parvenus à coder en **Ada** deux versions efficaces du routeur (liste chaînée et arbre binaire) et nous avons eu un regard critique sur chacune des versions.

2 Difficultés rencontrées

3 Nos expériences personnelles

3.1 Samy

En ce qui me concerne, je trouve que ce projet fut un très bon moyen pour mettre en pratique toutes les notions de programmation impérative acquises durant ce premier semestre à l'ENSEEIH, notamment, la manipulation de pointeurs, de listes chaînées et d'arbres binaires. Bien que le travail de groupe impose des contraintes en termes d'organisation, cette expérience fut tout de même enrichissante. Nous sommes parvenus à répartir les tâches et à planifier des séances de travail ensemble.

3.2 Maxime

Pour ma part, ce projet m'a dans un premier temps permis d'appliquer la notion de LCA vu dans un TP précédent, ainsi qu'un mini-projet. Il a ensuite été enrichissant d'un point de vue de la découverte du travail de groupe sur un projet en programmation. De plus, il s'agit du premier rendu que j'édite en \LaTeX , une compétence selon moi indispensable dans l'environnement ingénieur.

3.3 Ewan

Ce projet a été très formateur que ce soit sur le plan purement informatique ou sur le plan relationnel. Grâce à ce projet, je me sens beaucoup plus à l'aise sur les arbres et de manière générale sur des algorithmes plutôt complexes. Le projet m'a entre autres appris à utiliser rigoureusement les pointeurs. Ce projet, certes n'a pas été de tout repos, mais m'a permis d'améliorer mes compétences en tant que futur ingénieur.

4 Remerciements

Merci à l'ensemble des enseignants pour leurs aides précieuses apportées durant les séances de projet.

Annexe A

Raffinages

1. Maxime (Étudiant 1) : FONCTIONNEMENT GÉNÉRAL DU ROUTEUR

```
-- On définit la procédure Lire_Ligne qui prend un fichier texte T ainsi
qu'une chaîne de caractères Ch. Elle enregistre la ligne courante du
fichier T dans Ch et passe à la ligne suivante de ce fichier T.
-- On définit de même la procédure Lire_Caractere qui prend une chaîne de
caractères Ch ainsi qu'un caractère c. Cette procédure enregistre le
premier caractère de la chaîne de caractères Ch dans c et supprime ce
premier caractère de Ch.
```

R0 : Modéliser le routeur

R1 : Comment « Modéliser le routeur » ?

Initialiser les options par défaut

Lire et modifier les options selon les arguments donnés dans la ligne de commande

Exécuter le programme de routage

R2 : Comment « Initialiser les options par défaut » ?

```
taille <-10 taille : OUT entier
```

```
politique <- "FIFO" politique : OUT chaîne de caractères
```

```
statistiques <- True statistiques : OUT booléen
```

```
nom_table <- "table.txt" nom_table : OUT chaîne de caractères
```

```
nom_paquets <- "paquets.txt" nom_paquets : OUT chaîne de caractères
```

```
nom_resultats <- "resultat.txt" nom_resultats : OUT chaîne de caractères
```

R2 : Comment « Lire et modifier les options et arguments de la ligne de commande » ?

Modifier les valeurs des options taille, politique, statistiques,
nom_table, nom_paquets, nom_resultats : IN

R2 : Comment « Exécuter le programme de routage » ?

```
-- On initialise des fichiers texte cache, table de résultats et
statistiques vides
Tab          Tab : IN T_LCA
Paq          Paq : IN fichier texte
Cache        Cache : OUT type du cache
Res <- ""    Res : OUT fichier texte
Stat         Stat : OUT type de la statistique
Ligne <- "Init"  Ligne : OUT chaîne de caractères
Fin <- False   Fin : OUT booléen
RÉPÉTER
  Lire_Ligne(Paq, Ligne)      Paq : IN OUT fichier texte  Ligne : IN OUT
  SELON Ligne DANS
    "table" =>  Afficher_Texte (Tab)      Tab : IN fichier texte
    "cache" =>  Afficher_Texte (Cache)    Cache : IN fichier texte
    "stat"  =>  Afficher_Texte (Stat)     Stat : IN fichier texte
    "fin"   =>  Fin <- True                Fin : IN OUT booléen
    AUTRES  =>  Associer le résultat au paquet lu -- paquet lu dans
                la variable Ligne
                Mettre à jour le cache
                Mettre à jour les statistiques
  FINSELON
JUSQU'À Ligne = "" ou Fin
```

2. Samy (Étudiant 2) : ROUTEUR_LL

R0 : Réaliser le routeur_LL

R1 : Comment réaliser le routeur_LL ?

Initialiser le cache

Créer la table de routage à partir du fichier txt

Traiter les instructions du fichier paquets

R2 : Comment traiter les instructions du fichier paquets ?

Effectuer les affichages demandés

Chercher une route

R3 : Comment chercher une route ?

```
Cherche_Route(Cache, Adresse_Paquet, Route_Presente, Route, Interf,
Politique, True, Date);
```

```
SI Route_Presente alors
```

```
    Ecrire l'interface de la route sur le fichier de résultats
```

```
SINON
```

```
    Cherche_Route(Table_Routage, Adresse_Paquet, Route_Presente, Route,
Interf, Politique, False, Date);
```

```
SI Route_Presente alors
```

```
    Ajouter_IP (Cache, Politique, Precision_Cache, Taille, Taille_Max,
Adresse_Paquet, Interf, Default_Cache, Date);
```

```
SINON
```

```
    Indiquer qu'aucune interface est trouvée
```

```
FINSI
```

```
FINSI
```

```
Cherche_Route(Table_Routage, Adresse_Paquet, Route_Presente, Route, Interf,
Politique, False, Date);
```

```
SI Route_Presente ALORS
```

```
    Ajouter_IP (Cache, Politique, Precision_Cache, Taille, Taille_Max,
Adresse_Paquet, Interf, Default_Cache, Date);
```

```
SINON
```

```
    RIEN
```

```
FINSI
```

R0 : Planter Cherche_Route

R1 : Comment planter Cherche_Route ?

```

Liste : in T_LCA
IP_A_Router : in T_Adresse_IP
Route_Presente : out Booléen
Route : in out T_Adresse_IP
Interface : out Chaîne de caractères
Politique : in Chaîne de caractères
Est_Cache : in Booléen - - Permet de savoir si la Liste est un cache ou une
table de routage
Date : in out Entier

```

```

Masque_Max : T_Adresse_IP;
Cellule_Temp : T_LCA;
Cellule_Route : T_LCA;

```

```

Trouver la meilleure route dans Liste
Mettre à jour la date et la fréquence d'utilisation de cette route

```

R2 : Comment trouver la meilleure route dans Liste ?

```

TANTQUE non Est_Vide(Cellule_Temp) faire
  SI ( (IP_A_Router and Cellule_Temp.Masque) = Cellule_Temp.Adresse ) et
  ( Cellule_Temp.Masque >= Masque_Max ) alors
    Route_Presente <- Vrai
    Route <- Cellule_Temp.Route
    Interface <- Cellule_Temp.Interface
    Cellule_Temp <- Cellule_Temp.Suivant
  SINON
    Cellule_Temp <- Cellule_Temp.Suivant
FIN TANTQUE

```

R2 : Comment mettre à jour la date et la fréquence d'utilisation de cette route ?

```

SI Route_Presente et Est_Cache alors
  Mettre à jour la date ou la fréquence suivant la politique
SINON
  RIEN
FIN SI

```

R3 : Comment mettre à jour la date ou la fréquence suivant la politique ?

```

SI Politique = "LRU" alors
  Cellule_Route.All.Date <- Date + 1
  Date <- Date + 1
SINONSI Politique = "LFU alors

```

```

    Cellule_Route.All.Frequence := Cellule_Route.All.Frequence + 1
SINON
    RIEN
Fin SI

```

R0 : Implanter la procédure Ajouter_IP

R1 : Comment implanter la procédure Ajouter_Route ?

```

Cache : in out T_LCA
Politique : in Chaîne de caractères
Precision_Cache : in Entier
Taille : in out Entier
Taille_Max : in Entier
Adresse : in T_Adresse_IP
Interf : in Chaîne de caractères
Default : in out Entier
Date : in out Entier

Temp_Route : T_Adresse_IP <- 0;
UN_OCTET : constante T_Adresse_IP - 2 ** 8;

SI Taille >= Taille_Max alors
    Supprimer (Cache, Politique) - - Permet de supprimer une adresse
    suivant Politique
    Mettre à jour Date et Taille
SINON
    RIEN
FIN SI
Construire la route à ajouter dans le cache
Enregistrer (Cache, Temp_Route, Creer_Masque(Temp_Route), Interf, 1,
Date);
Mettre à jour Date et Taille

```

R2 : Comment mettre à jour Date et Taille

```

Taille <- Taille - 1
Default <- Default + 1

```

R2 : Comment construire la route à ajouter dans le cache

```

Temp_Route <- Adresse - (Adresse MOD UN_OCTET ** Precision_Cache)

```

R0 : Implanter la procédure Supprimer_FIFO (LCA)

R1 : Comment implanter la procédure Supprimer_FIFO ?

```
Cache : in T_LCA
Supprimer le dernier élément de la liste
```

R2 : Comment supprimer le dernier élément de la liste ?

```
Temp_Cellule : T_LCA
Temp_Cellule <- Cache
TANTQUE Temp_Cellule^.Suivant /= Rien
    Temp_Cellule <- Temp_Cellule^.Suivant
FIN TANTQUE

Temp_Cellule <- Rien
```

R0 : Implanter la procédure Supprimer_LRU (LCA)

R1 : Comment implanter la procédure Supprimer_LRU ?

```
Cache : in T_LCA
Cellule_Min : T_LCA <- Cache
Temp_Cellule : T_LCA <- Cache
Date_Min : Date <- Cache^.Date
Supprimer la route avec Date la plus petite
```

R2 : Comment supprimer la route avec Date la plus petite ?

```
TANTQUE Temp_Cellule /= Rien faire
    Trouver la date minimale
FIN TANTQUE
Supprimer la cellule contenant la date minimale
```

R3 : Comment trouver la date minimale ?

```
SI Temp_Cellule^.Date < Date_Min alors
    Date_Min <- Temp_Cellule^.Date
    Cellule_Min <- Temp_Cellule
SINON
    RIEN
Temp_Celulle <- Temp_Cellule^.Suivant
```

R3 : Comment supprimer la cellule contenant la date minimale ?

Supprimer(Cache, Cellule_Min)

R0 : Implanter la procédure Supprimer_LFU (LCA)

R1 : Comment implanter la procédure Supprimer_LFU ?

```
Cache : in T_LCA
Cellule_Min : T_LCA <- Cache
Temp_Cellule : T_LCA <- Cache
Freq_Min : Entier <- Cache^.Date
Supprimer la route avec Date la plus petite
```

R2 : Comment supprimer la route avec Date la plus petite ?

```
TANTQUE Temp_Cellule /= Rien faire
    Trouver la fréquence minimale
FIN TANTQUE
Supprimer la cellule contenant la fréquence minimale
```

R3 : Comment trouver la fréquence minimale ?

```
SI Temp_Cellule^.Frequence < Freq_Min alors
    Freq_Min <- Temp_Cellule^.Frequence
    Cellule_Min <- Temp_Cellule
SINON
    RIEN
    Temp_Cellule <- Temp_Cellule^.Suivant
FINSI
```

R3 : Comment supprimer la cellule contenant la fréquence minimale ?

Supprimer(Cache, Cellule_Min)

3. Ewan (Étudiant 3) : ROUTEUR_LA

TYPE T_arbre est un pointeur vers T_cellule

```

TYPE T_cellule EST enregistrement
  adresse:T_adresse_IP
  masque:T_adresse_IP
  frequence:Entier
  date:Entier
  interface:Chaine de caractere
  Suivant_g:T_arbre
  Suivant_d:T_arbre
fin enregistrement

```

R0 : Router une adresse

R1 : Comment ajouter la route dans le cache

```

SI le cache possède l'adresse cache:T_arbre in adresse_a_router:T_adresse_IP
  ne rien faire
SINON
  incrementer la route discriminé dans le cache route_discri:in out
  T_adresse cache:in interface:in chaine de caractere
  masque:in T_adresse_IP
FINSI

```

R2 : Comment incrementer la route discriminé dans le cache ?

```

SI (le cache est plein) cache: in
  supprimer la route dans le cache la moins utilisé selon la politique LRU
  cache: in out
SINON
  Rien faire
  rendre la route discriminante Table:LCA in route_discri:in out
  Profondeur<-0
  Enregistrer la route_discriminante dans le cache l'adresse
  route_discri:in cache:in out interface:in chaine de caractere
  masque:in T_adresse_IP
FINSI

```

R3 : Comment savoir si le cache est plein ?

```

Taille←Tailles(cache) cache:in
SI Taille=capacité alors

```

```

    renvoyer True
SINON
    renvoyer false
FINSI

```

R4 : Comment calculer la taille du cache

```

SI cache est vide
    retourner 0
SI le cache contient une adresse
    retourner 1
SINON
    retourner (calculer la taille du cache^Suivant_d )+(calculer la taille
    du cache^Suivant_g)
FINSI

```

R3 : Comment supprimer le cache selon la politique LRU

```

Trouver l'adresse le moins utilisé
    cache:in  adresse_mu:out
supprimer l'adresse la moins utilisé
    cache:in out  adresse_mu:in

```

R4 : Comment Trouver l'adresse la moins utilisé

```

adresse_mu← 0
fréquence←Nombre le plus grand du langage (ADA)
trouver la plus petite fréquence      cache:in      fréquence: out
Parcourir le cache et modifier adresse_mu et fréquence
cache:in out  adresse_mu:in

```

R5 : Comment parcourir le cache sachant adresse_mu et fréquence

```

SI cache est vide alors      cache:in
    rien faire
SINONSI il contient une adresse      cache:in
    alors supprimer si il contient l'adresse
        cache:in out  adresse_mu:in
SINON
    appeler parcourir (cache^Suivant_g) cache^Suivant_g:in
        adresse_mu:in
    appeler parcourir (cache^Suivant_d)
        cache^Suivant_d:in  adresse_mu:in
FINSI

```

R5 : Comment trouver la plus petite frequence du cache ?

```

SI cache est vide
    retourner le plus grand nombre en ada
SINONSI cache possede une adresse
    retourner cache^frequence
SINON
    retourner minimum(cache^Suivant_g,cache^Suivant_d)  cache:in
FINSI

```

R3 : Comment Enregistrer la route_discriminante dans le cache

```

SI le cache est vide
    Creer une nouvelle cellule
    rentrer ladresse son masque linterface dans la nouvelle cellule
        cache:in out interface:in adresse:in masque:in
        interface:in chaine de caractere masque:in T_adresse_IP
SINONSI le cache possede une adresse
    modifier le cache en fonction du profondeur eme bit
        adresse:in cache:in out interface:in chaine de caractere
        masque:in T_adresse_IP
SINON
    parcourir le cache en fonction du profondeur eme Bit
        adresse:in cache:in interface:in chaine de caractere
        masque:in T_adresse_IP
FINSI

```

R4 : Comment modifier le cache en fonction du profondeur eme bit?

```

SI le profondeur eme bit de l'adresse et de la route vaut 1
    profondeur<-profondeur+1
    creer une nouvelle cellule a droite et copier la route dans celle ci
    modifier le cache^suivant_d en fonction du profondeur eme bit
        adresse:in cache^Suivant_d:in out
        interface:in chaine de caractere
        masque:in T_adresse_IP
SINONSI le profondeur eme bit de l'adresse et de la route vaut 0
    profondeur<-profondeur+1
    creer une nouvelle cellule a gauche et copier la route dans celle ci
    modifier le cache^suivant_g en fonction du profondeur eme bit
        adresse:in cache^Suivant_g:in out
        interface:in chaine de caractere
        masque:in T_adresse_IP
SINON
    creer une nouvelle cellule a droite et a gauche
    mettre a droite la route ou l'adresse qui a pour profondeur eme

```

```

    bit 1 et l'autre a gauche adresse:in cache:in out
    interface:in chaine de caractere
    masque:in T_adresse_IP

```

FINSI

R4 : Comment parcourir le cache en fonction du profondeur eme Bit?

```

SI le profondeur eme bit de l'adresse vaut 1
    profondeur<-profondeur+1
    Enregistrer la route_discriminante dans le cache^Suivant_d
        adresse:in cache^Suivant_d:in out
        interface:in chaine de caractere
        masque:in T_adresse_IP
SINONSI le profondeur eme bit de l'adresse vaut 0
    profondeur<-profondeur+1
    Enregistrer la route_discriminante dans le cache^Suivant_g
        adresse:in cache^Suivant_g:in out
        interface:in chaine de caractere
        masque:in T_adresse_IP

```

FINSI

R3 : Comment savoir si l'adresse a router est présente dans le cache?

```

SI le cache est vide
    renvoyer Faux
SINONSI le cache possede une adresse
    renvoyer cache^adresse=adresse_a_router
SINON
    parcourir le cache en fonction du profondeur eme Bit
        adresse:in cache:in

```

FINSI

R4 : Comment parcourir le cache en fonction du profondeur eme Bit?

```

SI le profondeur eme bit de l'adresse vaut 1
    profondeur<-profondeur+1
    savoir si l'adresse a router est présente dans le cache^Suivant_d
        adresse:in cache^Suivant_g:in
SI le profondeur eme bit de l'adresse vaut 0
    profondeur<-profondeur+1
    savoir si l'adresse a router est présente dans le cache^Suivant_g
        adresse:in cache^Suivant_g:in

```

FINSI

R0 : Donner la route correspondante a l'adresse dans le cache

R1 : Donner la route correspondante a l'adresse dans le cache?

```

route:T_adresse_IP
SI le cache est vide
    Rien faire
SINONSI le cache possede une adresse
    dire si la route correspond a l'adresse
        adresse:in cache:in route:in out
SINON
    parcourir le cache en fonction du profondeur eme Bit
        adresse:in cache:in route:in
FINSI

```

R2 : Comment parcourir le cache en fonction du profondeur eme Bit?

```

SI le profondeur eme bit de l'adresse vaut 1
    profondeur<-profondeur+1
    Donner la route correspondante a l'adresse dans le cache^Suivant_d
        adresse:in cache^Suivant_d:in
SI le profondeur eme bit de l'adresse vaut 0
    profondeur<-profondeur+1
    Donner la route correspondante a l'adresse dans le cache^Suivant_g
        adresse:in cache^Suivant_g:in
FINSI

```

R2 : comment dire si la route correspond a l'adresse?

```

SI (adresse et arbre^Masque)=arbre^Adresse
    Cache present<-Vrai
    route<-arbre^adresse
    interface<-arbre^interface
SINON
    Cache present<-Faux
FINSI

```

R0 : savoir si le cache est vide

R1 : Comment savoir si le cache est vide?

```

retourner cache=Rien

```

R0 : savoir si le cache contient une adresse

R1 : Comment savoir si le cache est vide?

retourner (cache^suivant_g=Rien) et (cache^suivant_d =Rien)

Différences avec le premier raffinage

Dans un premier temps, nous avons rendu une première version des raffinages pour ce projet. En effet, après la réalisation concrète du projet, nous nous sommes rendu compte que les raffinages initiaux étaient incomplets. C'est pour cette raison que nous les avons corrigé afin de les adapter aux programmes livrés.

Mise à jour du module p_routeur_la

Après test du routeur avec cache en arbre binaire, nous nous sommes rendus compte que la politique LRU ne fonctionnait pas correctement. Ce problème a été résolu avec une simple modification de la fonction parcourir dans p_routeur_la.

Annexe B

Architecture du routeur

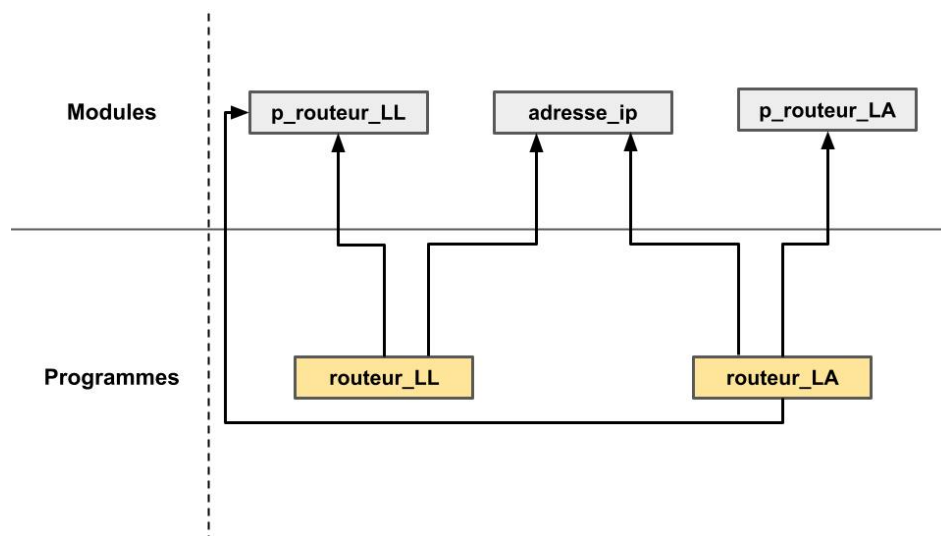


FIGURE B.1 – Architecture du routeur