Stockage et exploitation de tables de routage

Khadija Akkar ; Salma Elkhatri ; Taha Missouri

Janvier 2023

Professeur encadrant : Meryem Ouederni

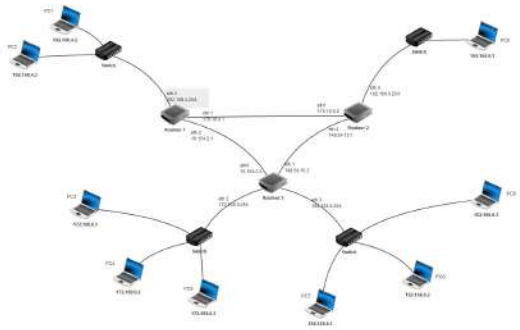
Résumé

Les tables de routage sont utilisées pour stocker et gérer les informations de routage pour les paquets dans un réseau. Les informations de routage contiennent des informations telles que la destination IP, le prochain saut (ou prochaine étape) pour atteindre cette destination, et le coût pour atteindre cette destination. Les tables de routage sont utilisées par les routeurs pour déterminer où envoyer les paquets réseau.

Table des matières

1 Introduction

L'exploitation et le stockage des tables de routage sont des aspects cruciaux pour assurer un fonctionnement optimal d'un réseau informatique. Les tables de routage contiennent des informations sur la destination des paquets et le chemin à suivre pour atteindre cette destination, permettant aux routeurs de déterminer où envoyer les paquets. Notre objectif était de trouver les chemins les plus optimaux et les moyens de maintenir et optimiser les tables de routage



2 Choix réalisés

2.1 Routeur simple

Ce premier code consiste à trouver l'interface de sortie à utiliser en fonction de la destination en gardant toutes les routes

Principaux Algorithme:

Définir la table de routage :

```
private
   type T_Route_Table;

   type T_Table is access T_Route_Table;

   type T_Route_Table is record
      Destination : T_Adresse_IP;
      Masque : T_Adresse_IP;
      Interface_T : Unbounded_String;
      Suivante : T_Table;
   end record;
```

remplir la table de routage :

```
procedure Remplire_Table(Fichier : in File_Type;
Table : in out T_Table) is
     UN_OCTET: constant T_Adresse_IP := 2 ** 8;
     IP : T_Adresse_IP;
     M : T_Adresse_IP;
     I : Unbounded_String;
  begin
     loop
        Get_IP(Fichier, IP);
        Get_IP(Fichier, M);
        I := Get_Line(Fichier);
        Enregistrer_Table(Table, IP, M, I);
     exit when End_Of_File(Fichier);
     end loop;

  end Remplire_Table;
```

Chercher et afficher l'interface convenable :

```
   procedure Donner_Resultats(Table : in out T_Table;
   paquets_txt  :  in File_Type) is
      UN_OCTET     : constant T_Adresse_IP := 2 ** 8;
      Resultats_txt: File_Type;
      IP           : T_Adresse_IP;
      T_Fichier    : Unbounded_String;
      P_Fichier    : Unbounded_String;
      R_Fichier    : Unbounded_String;
      i            : Integer;
      Stop         : Boolean;
      M_Trouvee : T_Adresse_IP;
      I_Trouvee : Unbounded_String;
   begin
      Analyser_L_Commande (T_Fichier, P_Fichier, R_Fichier);
      Create(Resultats_txt, Out_File, To_String(R_Fichier));
      begin
         i := 1;
         Stop := False;
         loop

            Commande_Paquets (paquets_txt, Stop, i, Table, IP);
            Chercher_Table(Table, IP, M_Trouvee, I_Trouvee);
            remplir fichier resultat;
            New_Line(Resultats_txt);
```

```
25            exit when End_Of_File(paquets_txt) or stop;
26            end loop;
27         exception
28            when End_Error =>
29               Put ("Blancs en surplus à la fin du fichier.");
30               null;
31         end;
32         Close (Resultats_txt);
33      exception
34         when E : others =>
35            Put_Line (Exception_Message (E));
36      end Donner_Resultats;
```

2.2 Routeur avec cache

Le routeur avec cache est un routeur simple avec cache.

2.2.1 Cache avec Listes lineaires

Principaux Algorithme:

```
1  function Chercher_Cache (Cache : in T_Cache_L; IP : in
2  T_Adresse_IP; M_Trouve_C : out T_Adresse_IP;Int_Trouve_C :
3  out Unbounded_String) return Boolean is
4        Cache0 : T_Cache_L;
5        Existe : Boolean;
6  begin
7     Existe := False;
8     Cache0 := Cache;
9     M_Trouve_C := 0;
10    while not Est_Vide_C(Cache0) loop
11      if (IP and Cache0.all.Masque) = Cache0.all.Destination then
12         if Comparer_Masque(M_Trouve_C, Cache0.all.Masque) then
13             M_Trouve_C := Cache0.all.Masque;
14             Int_Trouve_C := Cache0.all.Interface_T;
15         end if;
16         Existe := True;
17      else
18         null;
19      end if;
20      Cache0 := Cache0.all.Suivante;
21    end loop;
22    return Existe;
23 end Chercher_Cache;
```

```
procedure Vider_Cache_FIFO (Cache : in out T_Cache_L) is
      Cache0 : T_Cache_L;
   begin
      if not Est_Vide_C (Cache) then
         Cache0 := Cache;
         Cache := Cache.all.Suivante;
         Free(Cache0);
      else
         null;
      end if;
   end Vider_Cache_FIFO;

   procedure Vider_Cache_LFU (Cache : in out T_Cache_L) is
      Cache0 : T_Cache_L;
      Freq : Integer;
      IP_Supp : T_Adresse_IP;
   begin
      Cache0 := Cache;
      Freq := Cache0.all.Frequence;
      while Cache0 /= null loop
         if Cache0.all.Frequence < Freq then
            Freq := Cache0.all.Frequence;
            IP_Supp := Cache0.all.Destination;
         else
            null;
         end if;
         Cache0 := Cache0.all.Suivante;
      end loop;
      Supprimer_Cache(Cache, IP_Supp);
   end Vider_Cache_LFU;

   procedure Vider_Cache_LRU (Cache : in out T_Cache_L) is
      Cache0 : T_Cache_L;
      Time_use : Time;
      IP : T_Adresse_IP;
   begin
      Cache0 := Cache;
      Time_use := Clock;
      while Cache0 /= null loop
         if Cache0.all.Temps < Time_use then
            Time_use := Cache0.all.Temps;
            IP := Cache0.all.Destination;
            Cache0 := Cache0.all.Suivante;
         else
```

```
            Cache0 := Cache0.all.Suivante;
         end if;
      end loop;
      Supprimer_Cache(Cache, IP);

   end Vider_Cache_LRU;
```

2.2.2 Cache avec Arbres binaires

Principaux Algorithme:

```
procedure remp_cache(cache: in out T_cache_ptr;ip:T_IP_adresse;
masque:T_IP_adresse;intrface:Unbounded_String)  is
   C:T_IP_adresse:=ip;
   curseur:T_cache_ptr:=cache;
begin
   if cache=null then
      cache:=new T_cache;
      cache.destination:=1;
   end if ;
   while C/=0  loop
      if C mod 2=0 and C/2 /=0 then
          cache.left:=null;
          remp_cache(cache.left,ip,masque,intrface);
      elsif C mod 2 =1 and C/2 /=0 then
          cache.right:=null ;
          remp_cache(cache.right,ip,masque,intrface);
      elsif C/2 =0 then
         if C mod 2=0 then
             cache.left:=new T_cache'(ip,masque,intrface,clock
             ,null,null) ;
         elsif C mod 2=1 then
            cache.right:=new T_cache'(ip,masque,intrface,clock,
            null,null);
         end if ;
      end if ;
         C:=C/2;
   end loop;
end remp_cache;
```

```
function cher_cache(ip:in T_IP_adresse;cache:in T_cache_ptr)
return Unbounded_String is
              masque: T_IP_adresse:=0;
              inter:Unbounded_String;
              curseur:T_cache_ptr;
```

```
6               procedure cherch_cache(cache:in T_cache_ptr) is
7               begin
8                 if cache.destination /= 1 then
9                     if (ip and cache.masque) = cache.destination
10                    then
11
12                        if masque<cache.masque then
13                            masque:=cache.masque;
14                            inter:=cache.intrface;
15                            curseur:=cache;
16                        end if ;
17                    end if ;
18                elsif cache.destination=1 then
19                    cherch_cache(cache.right);
20                    cherch_cache(cache.left);
21                end if ;
22              end cherch_cache;
23    begin
24
25
26                curseur.temps:=clock;
27                return inter ;
28    end cher_cache;
29    procedure Free
30             is new Ada.Unchecked_Deallocation(Object =>
31             T_cache, Name => T_cache_ptr);
```

```
1     procedure vide_cache(cache:in out T_cache_ptr) is
2         time1:Time:=clock;
3         curseur:T_cache_ptr;
4     begin
5             if cache.destination/=1 then
6                 if cache.temps>time1 then
7                     curseur:=cache;
8                 end if ;
9             elsif cache.destination=1 then
10                vide_cache(cache.right);
11                vide_cache(cache.left);
12            end if ;
13
14            Free(curseur);
15    end vide_cache;
```

3 Difficultés rencontrés

La principale difficulté a été d'optimiser le programme que ce soit en temps ainsi qu'en mémoire. Ensuite il a aussi été très difficule de bien structurer les différentes procédure et fonction dans des modules cohérents. Faute de temps ce travail n'est pas correctement abouti. Il était difficile aussi de repérer et de résoudre les erreurs de codage, en raison de la complexité du code. C'était difficile aussi de se communiquer efficacement pour s'assurer que tout le monde est sur la même longueur d'onde. En conclusion il reste encore beaucoup de choses à faire malheureusement nous n'avons pas assez de temps pour terminé comme nous aimerions ce projet

4 Organisation d'équipe

Routeur simple : Salma El Khatri
Routeur LL : Khadija Akkar
Routeur LA : Taha Missouri