

# IP Address Lookup

~

## Structures et algorithmes

Destercq Lionel & Dubuc Xavier



10 novembre 2010

# Plan

- 1 Introduction
- 2 Algorithmes basés sur les arbres de préfixes
  - Arbres binaires (Binary Trie)
  - Arbres aux chemins compressés (Path-compressed)
- 3 Recherche binaire sur la longueur des préfixes
- 4 Schéma basé sur le hardware : TCAM
  - Ternary CAM
  - Réduction du coût mémoire
- 5 Un mot sur les lookups IPv6
  - Caractéristiques
  - Les TCAM pour IPv6
- 6 Références

# Introduction

Pour commencer, quelques définitions afin de bien situer notre sujet :

## Lookup

Le *lookup* d'une adresse IP est l'opération qu'effectue un routeur lorsqu'il consulte sa **table de forwarding** à la recherche de l'entrée correspondant à cette adresse IP afin d'obtenir l'interface de sortie sur laquelle il doit placer le paquet qu'il désire envoyer ainsi que le next-hop. Pour effectuer ce look-up, le routeur a recours à la méthode du **longest match prefix** qui sélectionne l'entrée de la table de forwarding qui a le plus de bits en commun avec le préfixe recherché.

# Exemple

**TABLE 2.1 Router's Forwarding Table Structure [1]**

Destination Address Prefix	Next Hop IP Address	Output Interface
24.40.32/20	192.41.177.148	2
130.86/16	192.41.177.181	6
208.12.16/20	192.41.177.241	4
208.12.21/24	192.41.177.196	1
167.24.103/24	192.41.177.3	4

**FIGURE:** Exemple de table de forwarding.

# CIDR et agrégation

## CIDR

Le plan d'adressage "*Classless Inter-Domain Routing*" permet plus de flexibilité dans l'adressage des adresses IPv4 mais il occasionne une expansion de la taille des tables de forwarding (*même si l'on peut contenir en partie cette expansion avec l'agrégation de préfixes IP mais pas dans tous les cas*).

## Exemple : Agrégation impossible

Imaginons qu'un client détenant le préfixe 208.12.21/24 change de provider sans changer de préfixe. Après ce changement, on ne peut plus agréger les préfixes allant de 208.12.16/24 à 208.12.31/24 car 208.12.21/24 n'est plus joignable via le même provider que les autres préfixes.

# Implémentation

Les **tables de forwarding** peuvent être implémentées de différentes manières, chacune de ces manières possédant ses avantages et ses inconvénients. Nous allons voir quelques-unes de ces méthodes et nous allons les comparer selon les critères suivants :

# Implémentation

Les **tables de forwarding** peuvent être implémentées de différentes manières, chacune de ces manières possédant ses avantages et ses inconvénients. Nous allons voir quelques-unes de ces méthodes et nous allons les comparer selon les critères suivants :

- **Vitesse de lookup** : *au moins assez pour suivre la fréquence d'entrée des paquets.*

# Implémentation

Les **tables de forwarding** peuvent être implémentées de différentes manières, chacune de ces manières possédant ses avantages et ses inconvénients. Nous allons voir quelques-unes de ces méthodes et nous allons les comparer selon les critères suivants :

- **Vitesse de lookup** : *au moins assez pour suivre la fréquence d'entrée des paquets.*
- **Exigence de stockage** : *moins de données stockées implique :*
  - accès mémoire rapides,
  - diminution de l'utilisation d'énergie.



# Implémentation

Les **tables de forwarding** peuvent être implémentées de différentes manières, chacune de ces manières possédant ses avantages et ses inconvénients. Nous allons voir quelques-unes de ces méthodes et nous allons les comparer selon les critères suivants :

- **Vitesse de lookup** : *au moins assez pour suivre la fréquence d'entrée des paquets.*
- **Exigence de stockage** : *moins de données stockées implique :*
  - accès mémoire rapides,
  - diminution de l'utilisation d'énergie.
- **Temps d'update** : *les routeurs doivent être capables de supporter 1000 BGP-Update par seconde.*

# Implémentation

Les **tables de forwarding** peuvent être implémentées de différentes manières, chacune de ces manières possédant ses avantages et ses inconvénients. Nous allons voir quelques-unes de ces méthodes et nous allons les comparer selon les critères suivants :

- **Vitesse de lookup** : *au moins assez pour suivre la fréquence d'entrée des paquets.*
- **Exigence de stockage** : *moins de données stockées implique :*
  - accès mémoire rapides,
  - diminution de l'utilisation d'énergie.
- **Temps d'update** : *les routeurs doivent être capables de supporter 1000 BGP-Update par seconde.*
- **Extensibilité** : *la taille de la table de forwarding peut augmenter de 25000 entrées par an.*

# Implémentation

Nous avons classés ces algorithmes en 3 catégories :

# Implémentation

Nous avons classés ces algorithmes en 3 catégories :

- ① les algorithmes basés sur les arbres de préfixes,
- ② les algorithmes de recherches binaires (ou par bisection),
- ③ les algorithmes/schémas basés sur le hardware.

# Plan

- 1 Introduction
- 2 Algorithmes basés sur les arbres de préfixes
  - Arbres binaires (Binary Trie)
  - Arbres aux chemins compressés (Path-compressed)
- 3 Recherche binaire sur la longueur des préfixes
- 4 Schéma basé sur le hardware : TCAM
  - Ternary CAM
  - Réduction du coût mémoire
- 5 Un mot sur les lookups IPv6
  - Caractéristiques
  - Les TCAM pour IPv6
- 6 Références

# Plan

- 1 Introduction
- 2 Algorithmes basés sur les arbres de préfixes
  - Arbres binaires (Binary Trie)
  - Arbres aux chemins compressés (Path-compressed)
- 3 Recherche binaire sur la longueur des préfixes
- 4 Schéma basé sur le hardware : TCAM
  - Ternary CAM
  - Réduction du coût mémoire
- 5 Un mot sur les lookups IPv6
  - Caractéristiques
  - Les TCAM pour IPv6
- 6 Références

## Définition de la structure

Il s'agit de la méthode la plus intuitive, on stocke les préfixes de la base de données dans un arbre binaire dont le parcours de chaque branche correspond à lire un 1 ou un 0. On choisit l'une de ces 2 branches en fonction du préfixe que l'on cherche à résoudre.

Prefix database

P1 \*  
 P2 1\*  
 P3 00\*  
 P4 101\*  
 P5 111\*  
 P6 1000\*  
 P7 11101\*  
 P8 111001\*  
 P9 1000011\*

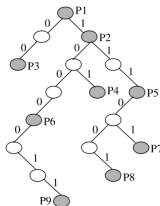


FIGURE: Exemple d'arbre binaire.

## Exemple de lookup

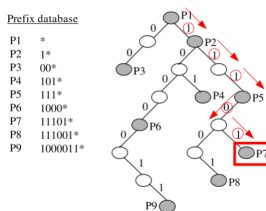


FIGURE: Exemple d'arbre binaire.

Si l'adresse de destination donnée est **11101000**, le **lookup** commence à la racine, puis se poursuit à droite, puis encore à droite, ... jusqu'à atteindre le noeud contenant le préfixe P7 qui est retourné comme résultat du lookup.

*(Ce noeud contient les informations de next-hop)*



## Exemple de lookup 2

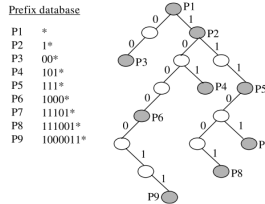


FIGURE: Exemple d'arbre binaire.

Si l'adresse de destination donnée est **111000**, que se passe-t-il ?

## Exemple de lookup 2

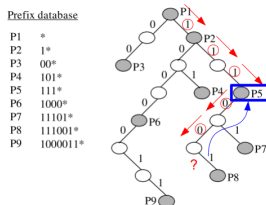


FIGURE: Exemple d'arbre binaire.

L'algorithme doit tenir en mémoire le dernier préfixe visité dans le cas où son chemin ne l'emmène pas vers un noeud contenant un préfixe.

## Complexités

Soit  $W$  la taille maximale d'un préfixe et  $N$  le nombre de préfixes contenu dans la table de forwarding,



- Intuitive, simple.
- Les complexités d'un **lookup** et d'un **update** en temps dans le pire des cas sont en  $O(W)$ .



- Dans le pire des cas, il faut **32 accès mémoire** par lookup.
- Dans le pire des cas, l'ajout d'un préfixe nécessite l'ajout de **32 noeuds**.
- La mémoire nécessaire est en  $O(N \times W)$ .

# Variantes

- 1 L'arbre peut être étendu à un arbre complet de telle sorte à ce que les préfixes soient tous stockés dans les feuilles, la structure devient en fait une structure plate.  
Si les adresses font 32bits,  $2^{32}$  entrées sont nécessaires !  
⇒ Impraticable.
- 2 Le "**leaf pushing**".

## Idées clés du Leaf pushing

- 1 Transformer un ensemble donné de préfixes en un ensemble de préfixes disjoints.
- 2 L'arbre correspondant comportera ses préfixes dans les feuilles et aucun dans les noeuds internes (*d'où le nom de la technique, les préfixes étant "poussés" vers les feuilles*).
- 3 Pour créer l'arbre, on ajoute une feuille à chaque noeud ne possédant qu'un seul fils et on y stocke le préfixe de l'ancêtre le plus proche. On supprime également tous les préfixes stockés dans les noeuds internes.

## Exemple de Leaf pushing

Prefix database

P1 \*  
 P2 1\*  
 P3 00\*  
 P4 101\*  
 P5 111\*  
 P6 1000\*  
 P7 11101\*  
 P8 111001\*  
 P9 1000011\*

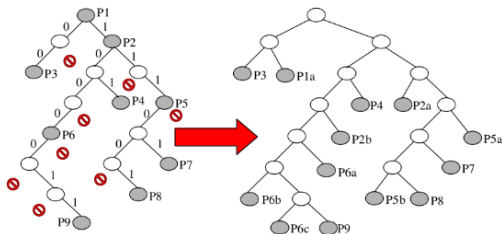


FIGURE: Exemple d'arbre "leaf pushed".

## Différences avec les arbres binaires

Les seules différences notables sont :

## Différences avec les arbres binaires

Les seules différences notables sont :

- ① on n'utilise pas la règle "*longest match prefix*" mais on trouve tout de même le préfixe le plus spécifique,
- ② certains préfixes doivent être stockés plusieurs fois  
⇒ augmentation de la mémoire nécessaire.



# Plan

- 1 Introduction
- 2 Algorithmes basés sur les arbres de préfixes
  - Arbres binaires (Binary Trie)
  - Arbres aux chemins compressés (Path-compressed)
- 3 Recherche binaire sur la longueur des préfixes
- 4 Schéma basé sur le hardware : TCAM
  - Ternary CAM
  - Réduction du coût mémoire
- 5 Un mot sur les lookups IPv6
  - Caractéristiques
  - Les TCAM pour IPv6
- 6 Références

# Introduction de la structure

L'idée de ces arbres est de réduire la hauteur en enlevant quelques noeuds internes et en stockant 2 infos supplémentaires par noeud restant : la **skip value** et **segment**.

## Skip value

Nombre indiquant le nombre de bits qui ont été passés sur le chemin.

## Segment

Chaîne de caractères de longueur variable indiquant la chaîne de digits passés sur le chemin.

# Exemple

Prefix database

P1 \*

P2 1\*

P3 00\*

P4 101\*

P5 111\*

P6 1000\*

P7 11101\*

P8 111001\*

P9 1000011\*

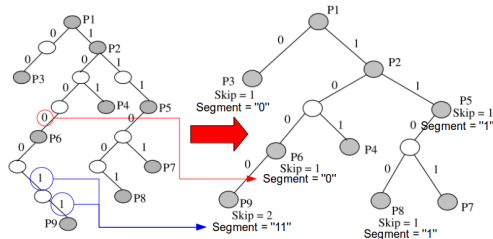


FIGURE: Exemple d'arbre "path compressed".

En **P9** la **skip value** est de 2 car on a passé les deux bits "1" sur le chemin vers ce noeud (**segment** est donc égal à "11", il faudra matcher cette chaîne pour atteindre **P9**).

# Complexités



Le stockage ne dépend plus de la longueur maximale de préfixe mais uniquement du nombre de noeuds de l'arbre, dénoté **N**.

⇒  $O(N)$



Dans le cas d'un arbre plein, l'arbre binaire n'est pas modifié.

⇒ **mêmes complexités**

# Plan

- 1 Introduction
- 2 Algorithmes basés sur les arbres de préfixes
  - Arbres binaires (Binary Trie)
  - Arbres aux chemins compressés (Path-compressed)
- 3 Recherche binaire sur la longueur des préfixes**
- 4 Schéma basé sur le hardware : TCAM
  - Ternary CAM
  - Réduction du coût mémoire
- 5 Un mot sur les lookups IPv6
  - Caractéristiques
  - Les TCAM pour IPv6
- 6 Références

# Introduction de la structure

On peut diviser l'opération du **longest match prefix** en une succession de *matching* **exacts**.

La structure de données utilisée est un ensemble de sous-tables  $H_1, H_2, \dots, H_W$  de telle sorte que  $H_i$  contienne tous les préfixes de taille  $i$ . Le matching va consister alors en une recherche dichotomique sur ces tables (et donc sur l'espace des longueurs de préfixe).

## Matching exact

Afin de réduire le temps pris par le *matching exact*, on utilise une **fonction de hashage** (*chaque sous-table utilise sa propre fonction*).

Pour simplifier, nous considérerons qu'il n'y a pas de **hash collision** (*2 préfixes hashés sur la même valeur*).

# Lookup

Le *lookup* commence par considérer d'abord la table  $H_{\frac{W}{2}}$ , de là 2 cas possibles :

- un noeud correspondant au préfixe est trouvé, dans ce cas on peut ignorer toutes les sous-tables  $H_i$  avec  $i < \frac{W}{2}$  (*longest matching prefix*), on continue donc la recherche dans les sous-tables  $H_j$  avec  $j > \frac{W}{2}$ .
- aucun noeud n'est trouvé, dans ce cas on peut ignorer toutes les sous-tables  $H_i$  avec  $i > \frac{W}{2}$ , on continue donc la recherche dans les sous-tables  $H_j$  avec  $j < \frac{W}{2}$ .

etc...



# Exemple

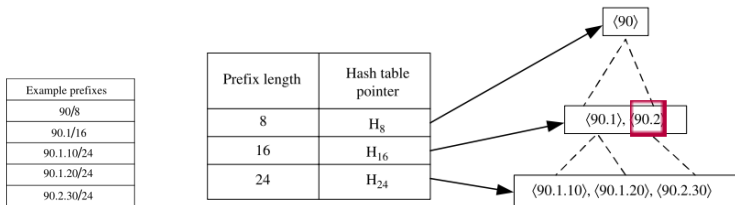


FIGURE: Exemple de structure.

$H_{16}$  contient un *marqueur*, c'est-à-dire un préfixe n'existant pas dans la **FIB**, il est utilisé pour aider à déterminer la branche à choisir lors de la recherche.

# Complexités

Soit  $W$  la taille maximale d'un préfixe,



- Update en  $O(\log_2 W)$
- Lookup en  $O(\log_2 W)$



- Mémoire nécessaire en  $O(N \log_2 W)$   
*(naïvement c'est en  $O(NW)$  car il peut y avoir jusqu'à  $W$  marqueurs par préfixe de la **FIB** mais au final seul  $\log_2 W$  d'entre eux sont nécessaires (ceux qui seront testés par l'algorithme))*

# Plan

- 1 Introduction
- 2 Algorithmes basés sur les arbres de préfixes
  - Arbres binaires (Binary Trie)
  - Arbres aux chemins compressés (Path-compressed)
- 3 Recherche binaire sur la longueur des préfixes
- 4 **Schéma basé sur le hardware : TCAM**
  - Ternary CAM
  - Réduction du coût mémoire
- 5 Un mot sur les lookups IPv6
  - Caractéristiques
  - Les TCAM pour IPv6
- 6 Références

# Plan

- 1 Introduction
- 2 Algorithmes basés sur les arbres de préfixes
  - Arbres binaires (Binary Trie)
  - Arbres aux chemins compressés (Path-compressed)
- 3 Recherche binaire sur la longueur des préfixes
- 4 **Schéma basé sur le hardware : TCAM**
  - **Ternary CAM**
  - Réduction du coût mémoire
- 5 Un mot sur les lookups IPv6
  - Caractéristiques
  - Les TCAM pour IPv6
- 6 Références

# Ternary CAM

- **Ternary Content-Adressable Memory**
- C'est un type de mémoire utilisée pour des recherches à haute vitesse.  
Plus connu en tant que mémoire associative.

# Ternary CAM

- **Ternary Content-Adressable Memory**
- C'est un type de mémoire utilisée pour des recherches à haute vitesse.  
Plus connu en tant que mémoire associative.
- Dispose de 3 états de matching :
  - OK
  - KO
  - X (valeur "don't care")

# Ternary CAM

- Ternary **C**ontent-**A**dressable **M**emory
- C'est un type de mémoire utilisée pour des recherches à haute vitesse.  
Plus connu en tant que mémoire associative.
- Dispose de 3 états de matching :
  - OK
  - KO
  - X (valeur "don't care")

## Exemple 1

10XX1 'match' les mots 10001, 10011, 10101, 10111.

# TCAM : Fonctionnement

TCAM sauve chaque entrée comme un couple (valeur, mask).  
La valeur mask est utile car elle sert à savoir si un bit est à considérer ou non dans la comparaison.



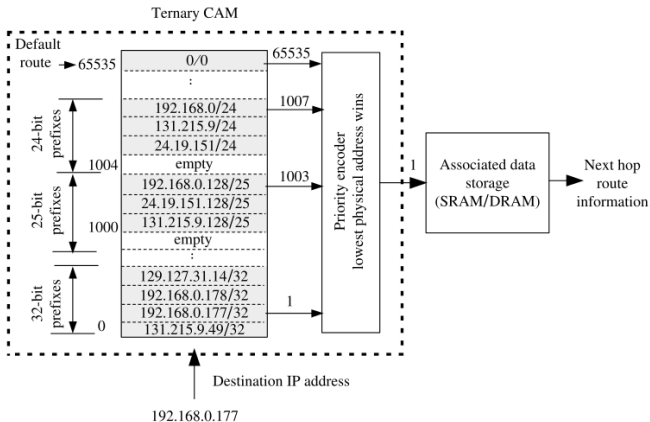
## TCAM : Fonctionnement

TCAM sauve chaque entrée comme un couple (valeur, mask).  
La valeur mask est utile car elle sert à savoir si un bit est à considérer ou non dans la comparaison.

### Exemple 2

Pour un mot de taille = 6, le préfixe 110\* est noté comme étant le couple (110000,111000)

## TCAM : dans un routeur



**FIGURE:** Schéma d'une table de forwarding utilisant TCAM

# TCAM : dans un routeur

Le fonctionnement est semblable :

- Comparaison de l'adresse IP destination avec toutes les entrées (bit à bit et simultanément)

## TCAM : dans un routeur

Le fonctionnement est semblable :

- Comparaison de l'adresse IP destination avec toutes les entrées (bit à bit et simultanément)
- Plusieurs matchs possibles :
  - Notion de priorité selon la localisation dans la mémoire
  - Entrées classées selon les longueurs de préfixes

Ainsi le match dont le préfixe est le plus grand est sélectionné.

## TCAM : dans un routeur

- Espaces de la mémoire vide en cas d'ajout de préfixes.
- Route par défaut :
  - Stockée au début de la mémoire
  - Valeur du mask = 0
  - Empruntée seulement s'il n'y a aucun autre match

# TCAM : performances



- TCAM n'a besoin que d'un accès mémoire pour son lookup
- Souvent utilisé car assez simple d'implémentation
- ~133 millions de lookup par seconde

## TCAM : performances



- TCAM n'a besoin que d'un accès mémoire pour son lookup
- Souvent utilisé car assez simple d'implémentation
- ~133 millions de lookup par seconde



Néanmoins, TCAM souffre d'un haut ratio coût/densité et d'une consommation énergétique assez élevée (10-15W/puce).

→ On va tenter de réduire la taille de la TCAM.

# Plan

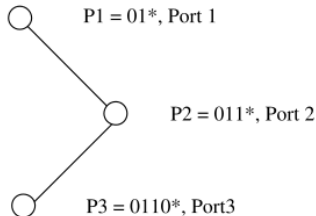
- 1 Introduction
- 2 Algorithmes basés sur les arbres de préfixes
  - Arbres binaires (Binary Trie)
  - Arbres aux chemins compressés (Path-compressed)
- 3 Recherche binaire sur la longueur des préfixes
- 4 **Schéma basé sur le hardware : TCAM**
  - Ternary CAM
  - **Réduction du coût mémoire**
- 5 Un mot sur les lookups IPv6
  - Caractéristiques
  - Les TCAM pour IPv6
- 6 Références



## Prefix Pruning

Littéralement "élagage de préfixe" ; on va tenter de supprimer les informations redondantes.

Mise en situation :



**FIGURE:** Exemple de préfixes redondants.

## Prefix Pruning

Si on suppose que P1 et P2 ont la même information de forwarding (ie. P1 et P2 forwardent sur le même port), alors P2 est redondant.

→ pour un longest match prefix qui devait se terminer en P2 elle se terminera en P1 si P2 est 'élagué'.

En réalité, les tables de forwarding ont un certain nombre de préfixes redondants.

→ on élague et donc on réduit les entrées.

~20-30 % de préfixes redondants

# Mask Extension

On va utiliser la propriété principale des TCAM :

- Jusqu'à présent :
  - Uniquement utilisées pour des préfixes (ce qui implique ?)
  - Voir le second exemple


# Mask Extension

On va utiliser la propriété principale des TCAM :

- Jusqu'à présent :
  - Uniquement utilisées pour des préfixes (ce qui implique ?)
  - Voir le second exemple
- Maintenant :
  - On veut étendre le type 'prefix-match' à un type 'ternary-match'
  - Voir le premier exemple

## Mask Extension

On va maintenant voir par un exemple comment passer d'un prefix-match à un ternary-match :



#	Prefix	Mask	Next hop port
P <sub>1</sub>	10011100	11111100	7
P <sub>2</sub>	10001100	11111100	7
P <sub>3</sub>	11011100	11111100	7
P <sub>4</sub>	10001000	11111000	5
P <sub>5</sub>	11010000	11110000	4
P <sub>6</sub>	11110000	11110000	7
P <sub>7</sub>	10010000	11110000	4

#	Prefix	Mask	Next hop port
P <sub>1</sub> & P <sub>2</sub>	10001100	11101100	7
P <sub>1</sub> & P <sub>3</sub>	10011100	10111100	7
P <sub>4</sub>	10001000	11111000	5
P <sub>5</sub> & P <sub>7</sub>	10010000	10110000	4
P <sub>6</sub>	11110000	11110000	7

FIGURE: Exemple de table de forwarding.

## Performances

Grâce à ces deux techniques, typiquement, la taille des tables de forwarding est de  $\sim 50\text{-}60\%$  par rapport à celle des tables sans optimisation.

## Performances

Grâce à ces deux techniques, typiquement, la taille des tables de forwarding est de  $\sim 50\text{-}60\%$  par rapport à celle des tables sans optimisation.

Conséquences de ces deux techniques :

- Prefix Pruning : aucun problème

## Performances

Grâce à ces deux techniques, typiquement, la taille des tables de forwarding est de  $\sim 50\text{-}60\%$  par rapport à celle des tables sans optimisation.

Conséquences de ces deux techniques :

- Prefix Pruning : aucun problème
- Mask Extension :
  - augmente la complexité des mises à jours
  - beaucoup de préfixes sont associés avec d'autres



# Performances


Grâce à ces deux techniques, typiquement, la taille des tables de forwarding est de  $\sim 50\text{-}60\%$  par rapport à celle des tables sans optimisation.

Conséquences de ces deux techniques :

- Prefix Pruning : aucun problème
- Mask Extension :
  - augmente la complexité des mises à jours
  - beaucoup de préfixes sont associés avec d'autres  $\rightarrow$  cela constitue un obstacle aux mises à jour incrémentales.

## Pruning & Mask Extension : Exemple

#	Prefix	Mask	Next hop port
P <sub>1</sub>	10011000	11111100	4
P <sub>2</sub>	11111100	11111100	7
P <sub>3</sub>	10001100	11111100	5
P <sub>4</sub>	10001000	11111100	5
P <sub>5</sub>	11010000	11110000	4
P <sub>6</sub>	11110000	11110000	7
P <sub>7</sub>	10010000	11110000	4



#	Prefix	Mask	Next hop port
P <sub>1</sub>	100110*		4
P <sub>2</sub>	111110*		7
P <sub>3</sub>	100011*		5
P <sub>4</sub>	10001*		5
P <sub>5</sub>	1101*		4
P <sub>6</sub>	1111*		7
P <sub>7</sub>	1001*		4

#	Prefix	Mask	Next hop port
P <sub>3</sub> & P <sub>4</sub>	10001000	11111100	5
P <sub>1</sub> & P <sub>7</sub>	10010000	11110000	4
P <sub>2</sub> & P <sub>6</sub>	11110000	11110000	7
P <sub>5</sub>	11010000	11110000	4

**FIGURE:** Exemple de réduction d'une table.

## Pruning & Mask Extension : Exemple

#	Prefix	Mask	Next hop port
$P_3 \& P_4$	10001000	11111000	5
$P_1 \& P_7$	10010000	11110000	4
$P_2 \& P_6$	11110000	11110000	7
$P_5$	11010000	11110000	4



#	Prefix	Mask	Next hop port
$P_3 \& P_4$	10010000	11111000	5
$P_1 \& P_5 \& P_7$	10010000	10110000	4
$P_2 \& P_6$	11110000	11110000	7

FIGURE: Exemple de réduction d'une table.

# Plan

- 1 Introduction
- 2 Algorithmes basés sur les arbres de préfixes
  - Arbres binaires (Binary Trie)
  - Arbres aux chemins compressés (Path-compressed)
- 3 Recherche binaire sur la longueur des préfixes
- 4 Schéma basé sur le hardware : TCAM
  - Ternary CAM
  - Réduction du coût mémoire
- 5 **Un mot sur les lookups IPv6**
  - Caractéristiques
  - Les TCAM pour IPv6
- 6 Références

# Plan

- 1 Introduction
- 2 Algorithmes basés sur les arbres de préfixes
  - Arbres binaires (Binary Trie)
  - Arbres aux chemins compressés (Path-compressed)
- 3 Recherche binaire sur la longueur des préfixes
- 4 Schéma basé sur le hardware : TCAM
  - Ternary CAM
  - Réduction du coût mémoire
- 5 **Un mot sur les lookups IPv6**
  - **Caractéristiques**
  - Les TCAM pour IPv6
- 6 Références

## IPv6 : Rappel

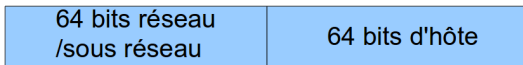


FIGURE: Structure d'une adresse IPv6

- les adresses seront données par préfixe /48 (une très grosse organisation pourra éventuellement avoir des /47 ou plus petit encore voire plusieurs /48),
- les /64 seront donnés lorsqu'il sera certain qu'un seul sous-réseau sera nécessaire,
- les /128 seront données lorsqu'il sera certain qu'un seul appareil sera nécessaire.

## Caractéristiques intéressantes

Des informations précédentes, on peut tirer les caractéristiques suivantes :

- ❶ **Aucun préfixe** de longueur comprise entre **64** et **128** bits ne sera utilisé.

## Caractéristiques intéressantes

Des informations précédentes, on peut tirer les caractéristiques suivantes :

- ❶ **Aucun préfixe** de longueur comprise entre **64** et **128** bits ne sera utilisé.
- ❷ La plupart des préfixes seront des **/48** et, dans une moindre mesure des **/64**.



## Caractéristiques intéressantes

Des informations précédentes, on peut tirer les caractéristiques suivantes :

- ❶ **Aucun préfixe** de longueur comprise entre **64** et **128** bits ne sera utilisé.
- ❷ La plupart des préfixes seront des **/48** et, dans une moindre mesure des **/64**.
- ❸ Très peu de préfixes **/128** (*tout comme les /32 pour IPv4*).

# Plan

- 1 Introduction
- 2 Algorithmes basés sur les arbres de préfixes
  - Arbres binaires (Binary Trie)
  - Arbres aux chemins compressés (Path-compressed)
- 3 Recherche binaire sur la longueur des préfixes
- 4 Schéma basé sur le hardware : TCAM
  - Ternary CAM
  - Réduction du coût mémoire
- 5 Un mot sur les lookups IPv6
  - Caractéristiques
  - Les TCAM pour IPv6
- 6 Références

## IPv6 : TCAM

Les TCAM supportent des mots de longueurs 36, 72, 144, 288 et 576 bits.

On est tenté d'utiliser des mots de longueurs = 144 bits.

Cependant, on a vu que dans IPv6 beaucoup de préfixes ont une longueur inférieure à 64 bits

## IPv6 : TCAM

Les TCAM supportent des mots de longueurs 36, 72, 144, 288 et 576 bits.

On est tenté d'utiliser des mots de longueurs = 144 bits.

Cependant, on a vu que dans IPv6 beaucoup de préfixes ont une longueur inférieure à 64 bits

→ plus de 50% d'un mot stockera la valeur 'don't care'.

On va présenter une approche afin de gagner 40% d'espace.

# IPv6 : TCAM

Dans cette approche :

- les préfixes sont divisés en deux groupes,  $G_S$  et  $G_L$ .
  - $G_S$  contient les préfixes avec moins de 72 bits.
  - $G_L$  le reste

## IPv6 : TCAM

Dans cette approche :

- les préfixes sont divisés en deux groupes,  $G_S$  et  $G_L$ .
  - $G_S$  contient les préfixes avec moins de 72 bits.
  - $G_L$  le reste
- Les blocs TCAM sont divisés en deux partitions,  $P_S$  et  $P_L$ 
  - Les préfixes dans  $G_S$  sont stockés dans  $P_S$
  - Les préfixes dans  $G_L$  sont stockés dans  $P_S$  (les 72 premiers bits) et servent de marqueur et dans  $P_L$ , les bits restants concaténés à un tag de 16 bits.

# IPv6 : TCAM

Dans cette approche :

- les préfixes sont divisés en deux groupes,  $G_S$  et  $G_L$ .
  - $G_S$  contient les préfixes avec moins de 72 bits.
  - $G_L$  le reste
- Les blocs TCAM sont divisés en deux partitions,  $P_S$  et  $P_L$ 
  - Les préfixes dans  $G_S$  sont stockés dans  $P_S$
  - Les préfixes dans  $G_L$  sont stockés dans  $P_S$  (les 72 premiers bits) et servent de marqueur et dans  $P_L$ , les bits restants concaténés à un tag de 16 bits.
- Stockage sous forme d'un 6-uplet (préfixe, p, m, v, Next Hop, T) où p et m désignent si l'entrée est un préfixe, un marqueur ou les deux, v si le nexthop est valide ( $v=1$ ) ou pas ( $v=0$ ) et T un tag qui désigne un sous groupe.

## IPv6 : TCAM - Algorithme

- 1 Les 72 bits significatifs d'une adresse destination  $A$  sont extraites et on cherche avec ces 72 bits dans  $P_S$ .



## IPv6 : TCAM - Algorithme

- 1 Les 72 bits significatifs d'une adresse destination  $A$  sont extraites et on cherche avec ces 72 bits dans  $P_S$ .
- 2 Si le meilleur match n'est pas un marqueur, alors on a le next hop.

## IPv6 : TCAM - Algorithme

- 1 Les 72 bits significatifs d'une adresse destination  $A$  sont extraites et on cherche avec ces 72 bits dans  $P_S$ .
- 2 Si le meilleur match n'est pas un marqueur, alors on a le next hop.
- 3 Sinon on extrait les 56 bits restant de  $A$  et on recherche ces 56 bits concaténés avec le tag de 16 bits associé dans  $P_L$ .

## IPv6 : TCAM - Algorithme

- 1 Les 72 bits significatifs d'une adresse destination  $A$  sont extraites et on cherche avec ces 72 bits dans  $P_S$ .
- 2 Si le meilleur match n'est pas un marqueur, alors on a le next hop.
- 3 Sinon on extrait les 56 bits restant de  $A$  et on recherche ces 56 bits concaténés avec le tag de 16 bits associé dans  $P_L$ .
- 4 Si le résultat de la recherche est invalide ( $v = 0$ ), alors le paquet est forwardé au next hop trouvé à l'étape 2.

## IPv6 : TCAM - Algorithme

- 1 Les 72 bits significatifs d'une adresse destination  $A$  sont extraites et on cherche avec ces 72 bits dans  $P_S$ .
- 2 Si le meilleur match n'est pas un marqueur, alors on a le next hop.
- 3 Sinon on extrait les 56 bits restant de  $A$  et on recherche ces 56 bits concaténés avec le tag de 16 bits associé dans  $P_L$ .
- 4 Si le résultat de la recherche est invalide ( $v = 0$ ), alors le paquet est forwardé au next hop trouvé à l'étape 2.
- 5 Sinon, le paquet est forwardé en utilisant le next hop trouvé à l'étape 3.

## IPv6 : TCAM - Un exemple

Soit la table suivante dont les adresses sont sur 12 bits, 8 bits pour  $P_S$ , et le tag est sur 4 bits.

Considérons que l'on veuille chercher l'adresse destination 1011.0110.1000 dans cette table.

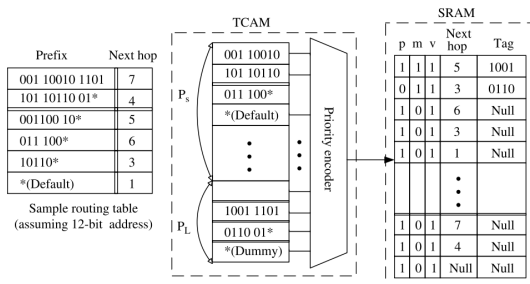


FIGURE: Exemple de table de forwarding.

## IPv6 : TCAM - Un exemple

Soit la table suivante dont les adresses sont sur 12 bits, 8 bits pour  $P_S$ , et le tag est sur 4 bits.

Considérons que l'on veuille chercher l'adresse destination 1011.0110.1000 dans cette table.

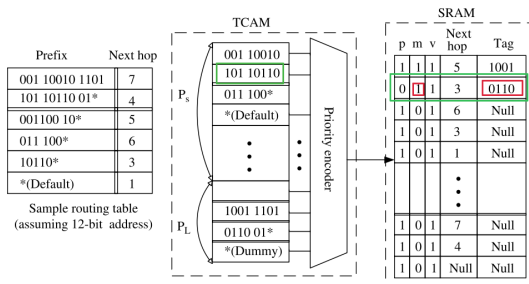


FIGURE: Exemple de table de forwarding.

# Plan

- 1 Introduction
- 2 Algorithmes basés sur les arbres de préfixes
  - Arbres binaires (Binary Trie)
  - Arbres aux chemins compressés (Path-compressed)
- 3 Recherche binaire sur la longueur des préfixes
- 4 Schéma basé sur le hardware : TCAM
  - Ternary CAM
  - Réduction du coût mémoire
- 5 Un mot sur les lookups IPv6
  - Caractéristiques
  - Les TCAM pour IPv6
- 6 **Références**

## Références (1/3)

- V. Fuller, T. Li, J. Yu, and K. Varadhan, “Classless inter-domain routing (CIDR) : an address assignment and aggregation strategy,” RFC 1519 (Proposed Standard), Sept. 1993. [Online]. Available at :  
[http ://www.ietf.org/rfc/rfc1519.txt](http://www.ietf.org/rfc/rfc1519.txt)
- D. Knuth, Fundamental Algorithms Vol. 3 : Sorting and Searching. Addison-Wesley, Massachusetts, 197
- V. Srinivasan and G. Varghese, “Faster IP lookups using controlled prefix expansion,” in Proc. ACM SIGMATICs, Madison, Wisconsin, pp. 1–10 (June 1998)
- D. R. Morrison, “PATRICIA - Practical algorithm to retrieve information coded in alphanumeric,” IEEE/ACM Transactions on Networking, vol. 17, no. 1, pp. 1093–1102 (Oct. 1968).



## Références (2/3)

- M. Waldvogel, G. Varghese, J. Turner, and B. Plattner, “Scalable high-speed IP routing lookups,” in Proc. ACM SIGCOMM, Cannes, France, pp. 25–36 (Sept. 1997).
- CYNSE10512 Network Search Engine, CYPRESS, Nov. 2002. [Online]. Available at : <http://www.cypress.com>
- Ultra9M – Datasheet from SiberCore Technologies. [Online]. Available at : <http://www.sibercore.com>
- P. Gupta, “Algorithmic search solutions : features and benefits,” in Proc. NPC-West 2003, San Jose, California (Oct. 2003).

## Références (3/3)

- H. Liu, “Routing table compaction in ternary CAM,” IEEE Micro, vol. 22, no. 1, pp. 58–64 (Jan. 2002)
- D. Pao, “TCAM organization for IPv6 address lookup,” in Proc. IEEE Int. Conf. on Advanced Communications Technology, Phoenix Park, South Korea, vol. 1, pp. 26–31 (Feb. 2005).