Réseaux II

Résumé blocus

Xavier Dubuc
Antoine Georis
Fabian Pijcke

UMONS
Université de Mons

9 janvier 2011

Table des matières

1 Introduction

1.1 Rappel des bases

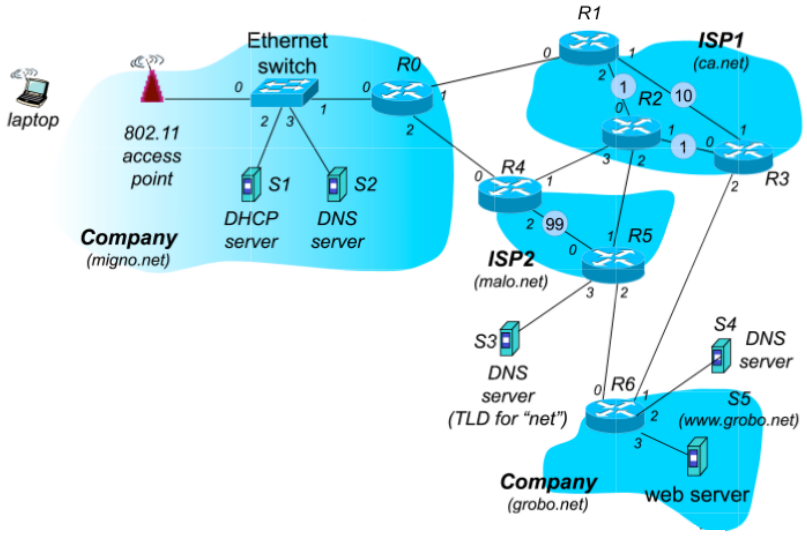


FIGURE 1 – Exemple

Imaginons que *laptop* cherche à joindre S_5 (*www.grobo.net*), voici les étapes successives se déroulant avant et pendant cet échange :

1. Lien avec une borne sans-fil
   - La *borne sans-fil* envoie à intervals réguliers des *flags* qui sont interceptés par le *laptop*. Un *flag* contient le *SSID(Service Set Identifier)* de la borne l'ayant émis.
   - le *laptop* envoie une *pro-request* à l'un des *SSID* qui lui sont disponibles. **Il se peut qu'il envoie directement une pro-request s'il connait déjà le SSID.**
   - La *borne sans-fil* répond via une *pro-réponse* (avec une phase d'authentification si le réseau est sécurisé) contenant, entre autre, le *BSSID (Basic Service Set Identifier)*. **Dans le cas d'une pro-request directe, le BSSID est "broadcast" même si l'on connaît le SSID.**
2. Obtention d'une adresse IP
   - Le *laptop* **broadcast** une demande d'adresse **IP** qui est acheminée au *serveur DHCP (Dynamic Host Configuration Protocol)* **("DHCP Discover")**. Lorsque la trame arrive à la *borne sans-fil* elle est transformée d'une trame 802.11 WiFi à une trame 802.3 Ethernet.

MAC SRC	BSSID	MAC DST	...	⇒	MAC SRC	MAC DST	...

TABLE 1 – 802.11 Vs 802.3

   Via ce message *DHCP Discover*, le *switch Ethernet* apprend l'adresse MAC du *laptop*. Il apprendra celle du *serveur DHCP* lorsque celui-ci répondra.
   - Le *serveur DHCP* répond en broadcastant une proposition d'adresse (broadcast nécessaire, pour que l'adresse ne soit pas donnée à 2 équipements différents). La table de switching du *switch Ethernet* est désormais :

ADDR MAC	INTERFACE OUT
MAC LAPTOP	0
MAC DHCP	2

TABLE 2 – Table de switching du *switch Ethernet*

   - Le *laptop* répond au *serveur DHCP* pour lui signifier qu'il accepte l'adresse.
   - Le *serveur DHCP* répond par une confirmation à *laptop*.

3. Accès au serveur DNS pour résoudre `www.grobo.net` *(Domain Name System)*
   On doit connaître l'adresse MAC du *serveur DNS* mais on ne connait que son adresse IP. On utilise alors le protocole **ARP** *(Address Resolution Protocol)* afin d'obtenir cette adresse et on ajoute la réponse reçue (donc l'adresse MAC du *serveur DNS*) dans la table de switching.
4. Résolution de `www.grobo.net`
   Dans notre cas, on va déléguer la tâche au *serveur DNS* (qui sert de cache en général) de manière itérative (comme en pratique) en lui demandant l'adresse IP correspondant à `www.grobo.net`. *(On suppose que S_2 connait S_3)*
   – le *serveur DNS* (S_2) demande au *TLD(Top Level Domain)* pour ".net" (S_3) s'il connaît `www.grobo.net`.
   – Celui lui répond qu'il devrait demander à S_4 qui lui répond qu'il connaît et qu'il est joignable à S_5.
5. Calcul du chemin jusque S_3 S_2 connaît de manière statique le lien vers R_0 pour joindre internet ($0.0.0.0/0$, préfixe par défaut).
   – inter-AS
   Via *BGP (Border Gateway Protocol)*, chaque routeur en bordure d'un réseau proprage la liste des préfixes qu'il peut atteindre ainsi que la liste des *AS (Autonomous System)* par lequel il faut passer pour atteindre une adresse dans chacun de ces préfixes *(AS-PATH)*.
   De cette manière, R_0 sait quel chemin est le plus court pour joindre S_3 (s'il y en a plusieures, il effectue un processus de décision pour en choisir un). R_0 passera donc par R_4.

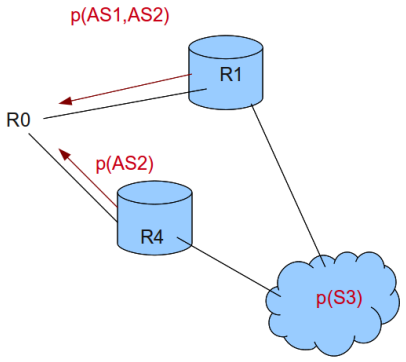


FIGURE 2 – Topologie et Messages BGP R_0, R_1 et R_4

   – intra-AS
   Il convient ici de considérer les poids des liens (via *OSPF (Open Shortest Path First)*
   Pour rappel, un routeur a 4 moyens d'apprendre une route (par ordre de priorité) :
   (a) de manière **statique** : rentrée par l'opérateur manuellement,
   (b) de manière **directe** : par la topologie du réseau (les cables connectés),
   (c) de manière **intra-domaine** via *OSPF*,
   (d) de manière **extra-domaine** via *BGP*.
   – S_2 contacte S_3 comme dit précédemment puis effectue le même raisonnement pour S_4.
   – S_2 contacte donc S_4. Via le routage, R_0 reçoit 2 routes d'$AS-PATH$ de même longueur ($AS3, AS1$ et $AS3, AS2$). Il effectue un "tie-break" c'est-à-dire qu'il fait un choix parce qu'il faut bien en faire un (en général il prend l'adresse **IP** la plus petite.
6. S_2 connaît à présent l'adresse **IP** de S_5 (qui correspond à `www.grobo.net`) et la donne au *laptop*. Celui-ci sait qu'elle n'est pas sur le réseau local, il effectue donc une requête **ARP** pour obtenir la *gateway*. En effet il connaît l'adresse **IP** de celle-ci grâce au *serveur DHCP* qui lui a communiqué les adresses **IP** des routeurs passerelles du réseau. La table de routage de *laptop* est donc comme suit :

Préfixe	Interface Out	Passerelle
$10.0.0/24$	$wlan0$	
$0.0.0.0/0$	$wlan0$	$p(R_0)$

TABLE 3 – Table de routage de *laptop*

7. S_2 envoie sa requête à S_5.

1.2 Traffic Engineering

Les chemins les plus courts sont calculés via **Dijkstra** ce qui assure que chaque paquet utilisera le plus court chemin pour être transmis. Le problème à ça c'est que le chemin le plus court sera le seul et unique utilisé. On peut avoir envie de répartir équitablement le trafic sur les 2 liens de sorties.

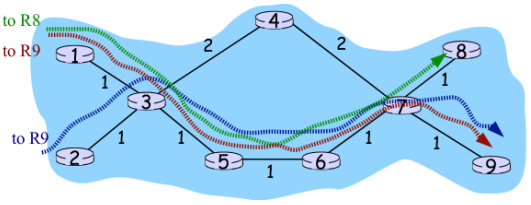


FIGURE 3 – Exemple d'emploi du plus court chemin à mauvais escient

On peut, pour résoudre à cela, par exemple, modifier les coûts du second chemin afin que les coûts des 2 chemins soient identiques. Il faut alors également modifier **Dijkstra** pour qu'il garde tous les chemins de coût minimum (et pas un seul). C'est l'idée du protocole *ECMP (Equal Cost Multiple Path)*. On choisit entre ces chemins de manière aléatoire ou avec le *round-robin*. La manière aléatoire n'est pas adaptée car elle n'est pas **TCP-friendly**, en effet, on ne peut assurer qu'un flux **TCP** restera sur le même chemin. Le *round-robin* consiste lui à assigner un des chemins à chaque flux **TCP** différent rencontré. Pour rappel, un flux **TCP** est identifié par le quadruplet $(ADD\ SRC,\ PORT\ SRC,\ ADD\ DST,\ PORT\ DST)$. En parallèle avec chaque lookup **IP** il faudra donc calculer une valeur de hashage pour le quadruplet afin de déterminer le chemin à utiliser.

Cette méthode fonctionnera bien s'il y a plusieurs petits flots (ne marche pas avec peu de gros flots) et si il y a beaucoup de chemins de même coût (totalement dépendant de l'assignation des coûts). La fonction de hashage doit être bien déterminée pour être assez discriminante pour ne pas assigner la route à flots très proches (mêmes couples d'adresses par exemple). De plus, si tous les routeurs utilisent la même fonction de hashage, cela crée une **polarisation** du réseau ; en effet, pour chaque routeur, on va choisir la même interface de sortie. *(On peut remédier à cette polarisation en ajouter le routeur ID dans la fonction de hashage)*

D'autres mécanismes, différents du *ECMP* existent. On peut modifier les poids des noeuds afin de minimiser le ratio charge/capacité pour chaque noeud en favorisant les chemins de même coût. Cette méthode requiert de connaître la matrice de trafic ce qui n'est pas toujours le cas. On peut aussi utiliser le routage explicite : "Tel flux prend tel chemin" ou encore un autre schéma de forwarding (comme **MPLS** *(Multiprotocol Label Switching)*) ou un protocole de signalisation pour installer un état (comme **RSVP-TE** *(Resource reSerVation Protocol -Traffic Engineering))*.

2 Introduction au routage inter-domaine

2.1 Objectifs fondamentaux

- Permettre de transmettre des paquets IP par le meilleur chemin lorsqu'il est nécessaire de passer par plusieurs domaines de transit (souvent, "meilleur chemin" revient à "chemin le moins cher")
- Prendre en compte les politiques de routage de chaque domaine
- Autonomie (à l'intérieur d'un domaine, les opérations sont effectuées indépendamment des autres, chaque domaine peut faire tourner son propre IGP)
- Cacher la topologie détaillée des domaines de transit (routage hiérarchique, nécessaire car le graphe inter-domaines possède des millions de noeuds)

2.2 Définitions

- Un système autonome (AS) est un réseau sous l'administration d'une entité unique (BELNET, Abilène, ...)
- Un préfixe est un ensemble d'adresses IP spécifié dans la notation CIDR (193.190.192/22 (UMons), ...)
- Lien privé : câble reliant deux routeurs appartenant à deux AS différents
- Connection via un point d'interconnexion : En général via un switch Gigabit (ou plus)

2.3 Politiques de routage

Il y a deux politiques principales :

Un lien client-vendeur (un client C achète de la connectivité internet à un provider P)

Un lien à coût partagé (les domaines X et Y se mettent d'accord pour échanger des paquets en utilisant un lien direct ou un point d'interconnexion)

Bien sûr, des politiques plus complexes sont possibles (et existent)

Matrice de politique de transit

		Vers		
		Provider	Pair	Client
De	Provider	Non	Non	Oui
	Pair	Non	Non	Oui
	Client	Oui	Oui	Oui

RPSL Pour faire appliquer ces règles (ou d'autres), un mécanisme de filtres est mis en place. Les administrateurs de réseaux peuvent donc appliquer un ou plusieurs filtres à chaque "porte" de leur réseaux : des filtres d'imports, qui permettent de limiter le trafic entrant, et des filtres d'export, qui permettent de limiter le trafic sortant.

Ces filtres sont en général exprimés dans un langage spécifique au vendeur du routeur, le standard est RPSL (Routing Policy Specification Language)

Exemples simples de syntaxe RPSL :

```
import: from AS# accept list_of_AS

import: from BELNET accept UMONS
import: from LEVEL3 accept ANY

export: to AS# announce list_of_AS

export: to UMONS announce ANY
export: to LEVEL3 announce UMONS UCLOUVAIN ...
```

3 BGP

Il s'agit du standard pour le routage inter-domaines

Permet à chaque AS ...

- ... de savoir si tel ou tel préfixe peut être atteint depuis les AS voisins
- ... de propager cette information dans chaque routeur interne à l'AS
- ... de déterminer les "bonnes" routes vers les sous-réseaux en se basant sur l'information d'atteignabilité et la politique de l'AS.

BGP permet également aux sous-réseaux de signaler leur existence.

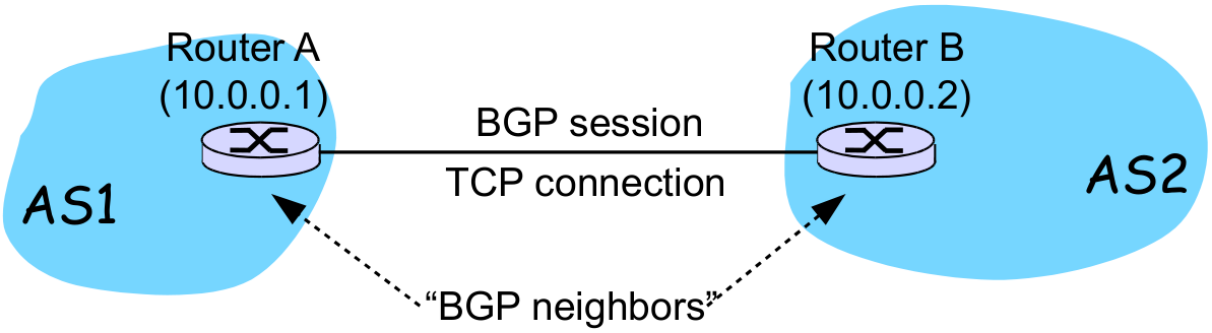
Pour remplir ces objectifs, BGP utilise un protocole de vecteurs de distance et des mises à jour incrémentales. (Avertisement de mise à jour seulement quand l'information change)

3.1 Sessions

Principe Les paires de routeurs BGP s'échangent des routes sur leur connexion TCP semi-permanente : la session BGP

Les routeurs disposant d'une session BGP sont appelés pairs BGP ou voisins BGP.

Le port par défaut pour les sessions BGP est 179.



Configuration d'une session BGP Préliminaire : les deux routeurs doivent être capables de s'atteindre

CISCO :

```
# router bgp [ASN]
# neighbor [IP_VOISIN] remote-as [ASN_VOISIN]
```

Juniper :

```
routing-options {
  autonomous-system [ASN];
}

protocols {
  bgp {
    group [NOM_GROUPE_VOISINS] {
      peer-as [ASN_VOISIN];
      type external;
      neighbor [IP_VOISIN];
    }
  }
}
```

3.2 Messages

4 grands types de messages dans le protocole BGP :
- OPEN (ouverture de session, éventuellement authentification)
- UPDATE (avertissement des chemins neufs / obsolètes)
- KEEPALIVE (maintient une connexion, ack du OPEN)
- NOTIFICATION (erreurs, fermeture de connexion)

Le header d'un message BGP est composé des champs suivants :
- Marker (16 bytes) inutilisé, ne doit contenir que des 1
- Longueur (2 bytes) longueur du paquet, header compris
- Type (1 byte) 1=OPEN, 2=UPDATE, 3=NOTIFICATION, 4=KEEPALIVE, 5=ROUTE REFRESH
- Corps du message

Message OPEN Le corps est composé des champs suivants :
- Version (1 byte) vaut 4 (00000100)
- ASN de l'émetteur (2 bytes)
- Temps d'attente (2 bytes)
- Identifiant BGP (4 bytes) Adresse IP d'une des interfaces du routeur émetteur
- Longueur des paramètres optionnels (1 byte)
- Paramètres optionnels (triplets [type, longueur, valeur])

Message UPDATE Le corps est composé des champs suivants :
- Nombre de routes obsolètes (2 bytes)
- Routes obsolète (variable)
- Longueur des attributs de chemins (2 bytes)

– Attributs de chemins (variable)
– Network Layer Reachability Information [NLRI] (variable)

3.3 Routes

Définition Une route est la combinaison d'un préfixe de destination et d'attributs de chemin, elle est transportée par un message UPDATE, le préfixe de destination est spécifié dans la partie NLRI du message et un routeur BGP annonce une seule route par préfixe de destination.

Avertissements de routes Ils proviennent de diverses sources :
– Routes statiques, configurées manuellement dans le routeur
– Redistribution de route (apprises par un protocole de routage intra-domaine, iBGP par exemple), mais c'est une mauvaise habitude vu que les instabilités intra-domaine seraient répercutées à l'extérieur et risqueraient de rendre le BGP global instable
– Apprentissage de routes depuis un autre routeur BGP (chaque noeud BGP avertit ses voisins de ses meilleures routes)

Abandon de routes Principe : quand une destination n'est plus atteignable, un routeur BGP doit en avertir ses voisins.

Les messages UPDATE sont utilisés pour informer qu'une route n'est plus disponible, on appelle ces messages WITHDRAW alors que, strictement, aucun message WITHDRAW n'existe dans la spécification BGP. Un WITHDRAW ne doit être envoyé que pour des routes précédemment annoncées.

Un WITHDRAW implicite est un message UPDATE modifiant juste un attribut de la route (c'est-à-dire que la destination peut toujour être atteinte, mais avec d'autres paramètres car il faut passer par d'autres routeurs BGP par exemple).

3.4 Attributs des chemins

Une route peut contenir plusieurs attributs ; les deux les plus importants sont NEXT-HOP (IP du prochain routeur BGP sur la route) et AS-PATH (liste des ASN des routeurs suivants sur la route)

L'AS-PATH, en plus de servir de métrique, permet d'éviter les boucles de routage : un routeur BGP voyant son propre ASN dans l'AS-PATH ne considérera pas cette route.

Attributs de chemins standardisés

Nom	Code	Implémentation obligatoire	attribut obligatoire	Transitif
ORIGIN	1	Y	Y	
AS-PATH	2	Y	Y	
NEXT-HOP	3	Y	Y	
MED (Multi Exit Discriminator)	4			
LOCAL_PREF	5	Y		
ATOMIC_AGGREGATE	6	Y		
AGGREGATOR	7			Y
COMMUNITIES	8			Y
ORIGINATOR_ID	9			
CLUSTER_LIST	10			
EXTENDED_COMMUNITIES	16			

```
Prefix | AS-Path | Origin | Next-hop | Local-Pref | MED | Communities | ...

6.1.0.0/16|668|IGP|198.32.8.202|100|0|668:100 11537:3000|NAG||
6.2.0.0/22|668|IGP|198.32.8.202|100|0|668:100 11537:3000|NAG||
6.3.0.0/18|668|IGP|198.32.8.202|100|0|668:100 11537:3000|NAG||
6.4.0.0/16|668|IGP|198.32.8.202|100|0|668:100 11537:3000|NAG||
6.5.0.0/19|668|IGP|198.32.8.202|100|0|668:100 11537:3000|NAG||
6.6.0.0/16|668|IGP|198.32.8.202|100|0|668:100 11537:3000|NAG||
6.8.0.0/20|668|IGP|198.32.8.202|100|0|668:100 11537:3000|NAG||
6.9.0.0/20|668|IGP|198.32.8.202|100|0|668:100 11537:3000|NAG||
6.10.0.0/15|668|IGP|198.32.8.202|100|0|668:100 11537:3000|NAG||
6.14.0.0/15|668|IGP|198.32.8.202|100|0|668:100 11537:3000|NAG||
8.6.244.0/23|11096 6356|IGP|198.32.155.193|200|0|11096:307 11096:501 11537:950|NAG||
8.10.208.0/24|10466 32554|IGP|198.32.8.199|260|0|11537:260 11537:910 11537:950|NAG||
9.4.0.0/16|20965 559|IGP|198.32.8.202|100|0|11537:2501 20965:155|NAG||
12.0.48.0/20|10578 1742|IGP|198.32.8.199|200|0|10578:800 10578:840 11537:950|NAG||
12.6.208.0/20|10578 1742|IGP|198.32.8.199|200|0|10578:800 10578:840 11537:950|NAG||
12.107.208.0/23|81 22753|IGP|198.32.8.202|200|0|11537:950 11537:2000|NAG||
12.144.59.0/24|10466 13778|IGP|198.32.8.199|260|0|11537:260 11537:950 11537:2000|NAG||
12.151.0.0/24|10466 11558|IGP|198.32.8.199|260|0|11537:260 11537:902 11537:950|NAG||
12.151.1.0/24|10466 11558|IGP|198.32.8.199|260|0|11537:260 11537:902 11537:950|NAG||
12.161.8.0/21|10466 88|IGP|198.32.8.199|260|0|11537:260 11537:950|NAG||
12.174.210.0/23|5661 21712|IGP|131.247.47.246|200|0|11537:902 11537:950|NAG||
18.0.0.0/8|10578 3|IGP|198.32.8.199|200|0|10578:800 10578:840 11537:950|NAG||
18.3.4.0/24|10578 3|INCOMPLETE|198.32.8.199|200|0|10578:800 10578:840 11537:950|NAG||
```

3.5 Machine à états finie

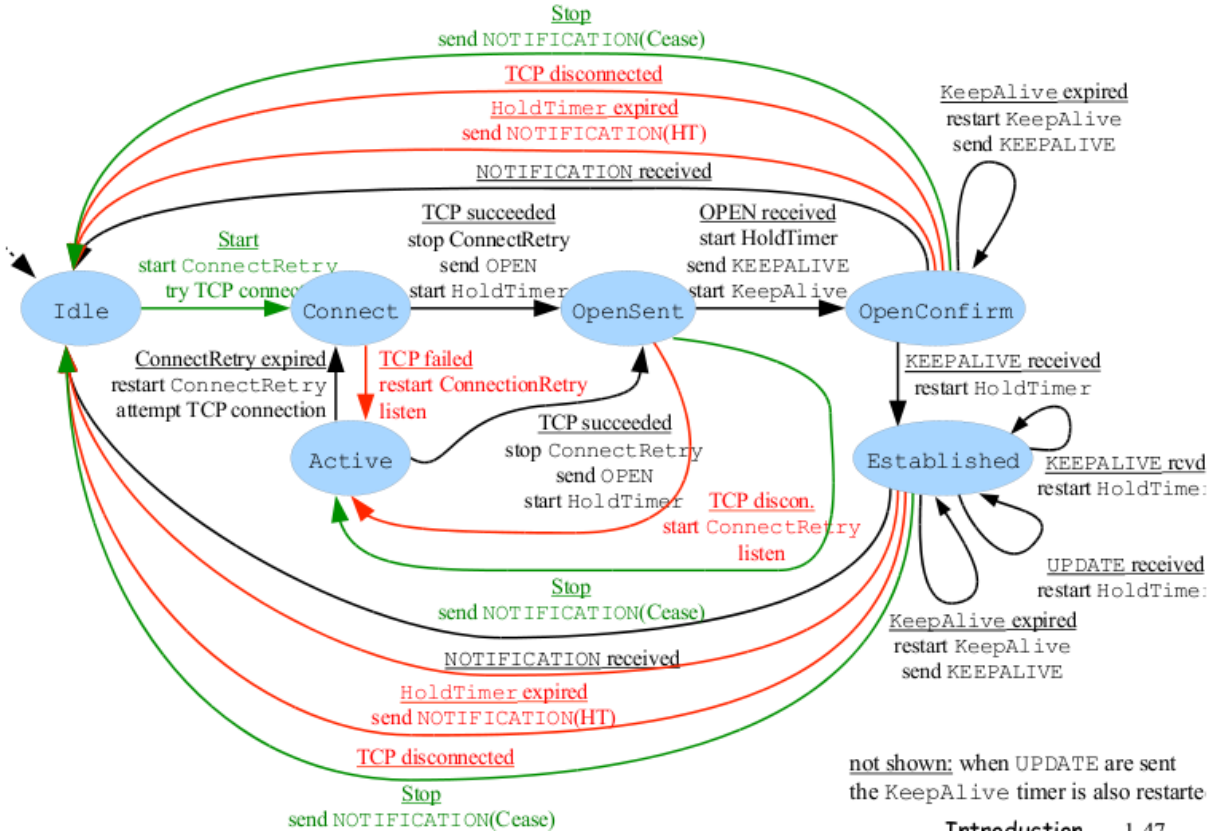
Le but de la machine à étatss BGP est de s'assurer de la gestion correcte des messages BGP comme de la robustesse des sessions BGP. Toute implémentation pour un routeur BGP doit suivre le même comportement que la machine à états.

La machine est composée de 6 états :

- Idle (état initial)
- Connect (tentative d'initiation de connexion TCP)
- Active (écoute de connexions TCP entrantes)
- OpenSent (OPEN envoyé, attente d'un OPEN entrant)
- OpenConfirm (OPEN reçu, KEEPALIVE envoyé, attente d'un KEEPALIVE entrant)
- Established (En marche, échange de KEEPALIVE et de UPDATE)

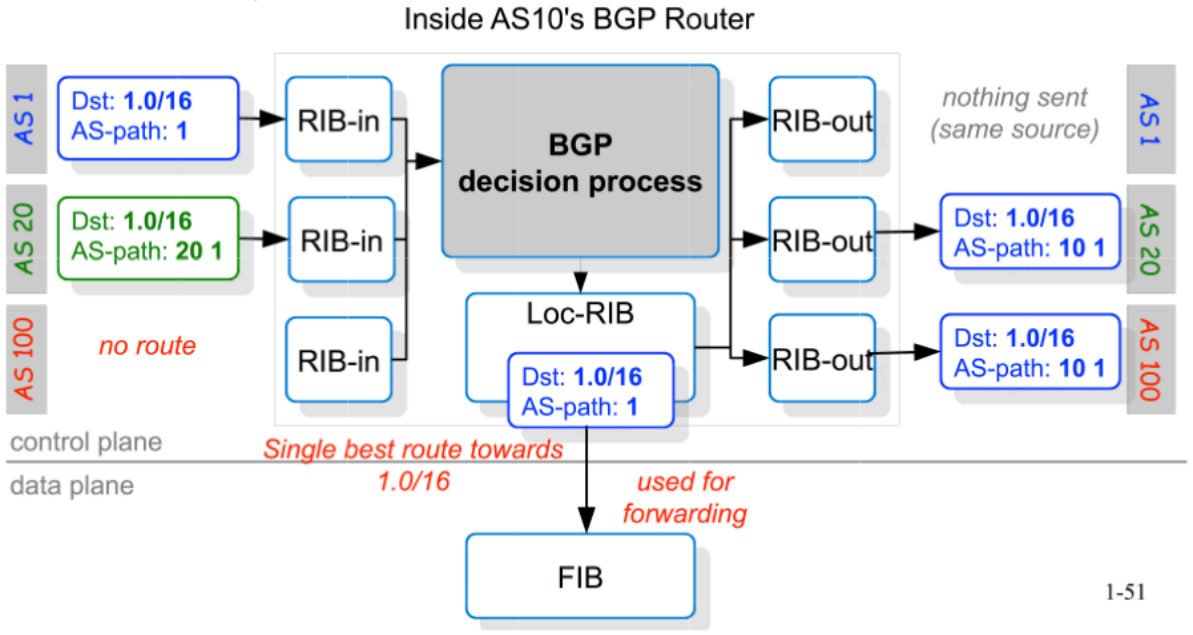
Et de 3 timers :

- ConnRetry (Espace les tentatives de connexion TCP)
- HoldTimer (Assure l'activité de la session BGP)
- KeepAlive (Envoie des KEEPALIVE régulièrement)



3.6 Processus de décision

Un routeur recevra souvent plusieurs routes pour joindre un même préfixe. Il doit alors choisir une unique meilleure route à communiquer à ses voisins.



Le processus de décision est composé d'une série de règles qui va réduire l'ensemble des routes admissibles pour un préfixe donné jusqu'à ce qu'il n'en reste qu'une, ce sera la meilleure route, qui sera ensuite communiquée aux voisins du routeur (en accord avec la politique de l'AS bien entendu).

Le processus simplifié ressemble à ceci :

- Ignorer si le NEXT-HOP n'est pas joignable
- Préférer les routes ayant un LOCAL-PREF supérieur
- Préférer les routes ayant un AS-PATH plus court

– Tie-break

Voici le processus de décision complet :

1. Ignorer si le NEXT-HOP n'est pas joignable
2. Préférer les réseaux originaires du réseau local
3. Préférer les routes ayant un LOCAL-PREF supérieur
4. Préférer les routes ayant un AS-PATH plus court
5. Préferer les ORIGIN les plus bas
6. Préférer les MED les plus bas
7. Préférer eBGP par rapport à iBGP
8. Préférer le NEXT-HOP le plus proche
9. Préférer le ROUTER-ID / ORIGINATOR-ID le plus bas
10. Préférer le CLUSTER-LIST le plus court
11. Préférer l'adresse du voisin la plus petite

On s'en doute, les trois derniers points sont des tie-breaks.

3.7 Filtres de routage

Objectifs

- Influencer le classement des routes
- Empêcher certaines routes d'être redistribuées à certains voisins
- Rejeter une route d'un voisin spécifique

Principes

- Ajouter des filtres d'import/export au modèle de routage BGP pour chaque voisin
- Filtres d'import (in-filters) : sélectionner les routes acceptables et changer leurs attributs
- Filtres d'export (out-filters) : sélectionner les routes redistribuables et changer leurs attributs
- Les filtres sont spécifiés dans un langage spécifique au vendeur du matériel

Un filtre est un ensemble de prédicats qui mènent à un ensemble d'actions.

Exemples de prédicats

- match destination prefix against IP prefix
- match next-hop against IP address / prefix
- test existence of an ASN in AS-PATH
- match AS-PATH against regular expression
- ...

Exemples d'actions

- Accept / reject the route
- Set / Increase / Decrease LOCAL-PREF
- Prepend AS-PATH
- Set MED
- Add / Remove community
- Remove private ASN from AS-PATH
- ...

Exemple d'application Un AS possède souvent plusieurs liens vers son provider. Un lien principal qui doit être utilisé autant que possible et un lien secondaire qui doit être utilisé lorsque le lien secondaire ne fonctionne pas.

On peut donc configurer un filtre donnant une priorité plus grande aux routes fournies par le routeur de la route principale

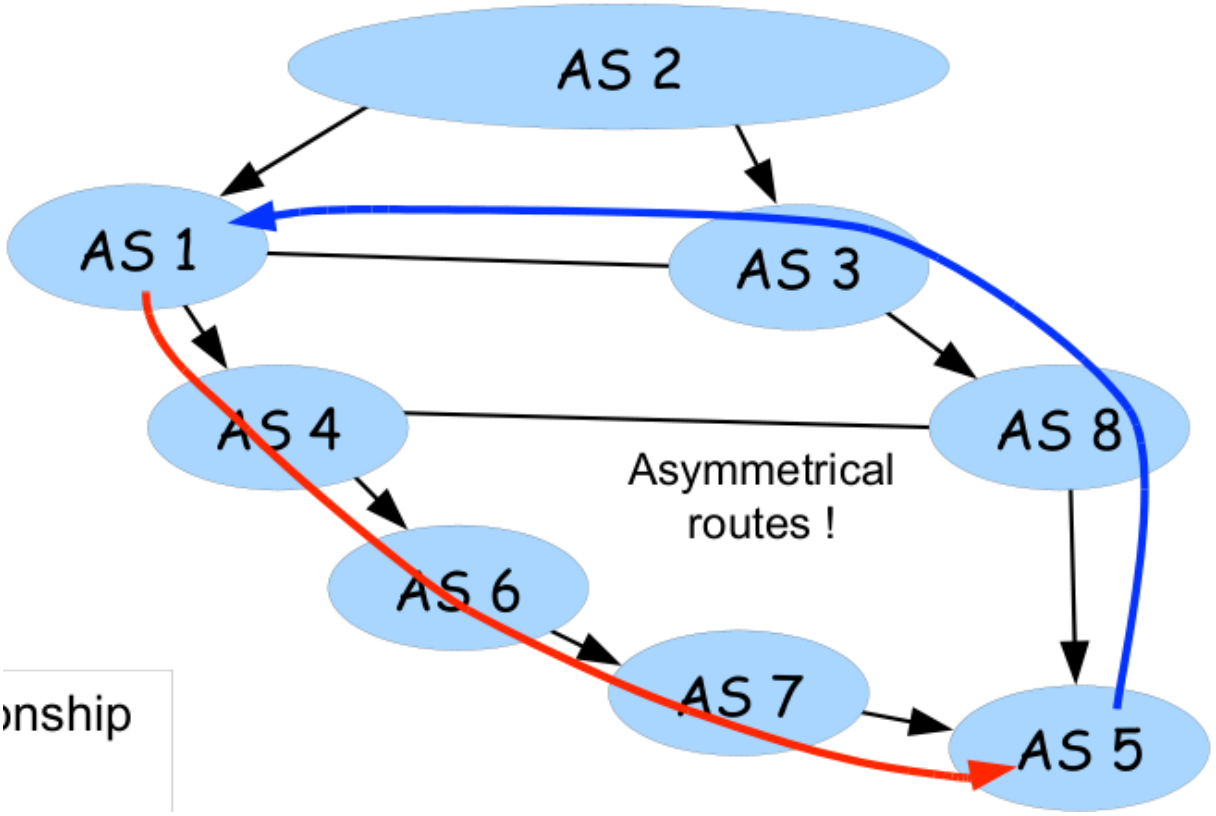
```
import: from AS2 RB at RA set localpref=200;
import: from AS2 RC at RA set localpref=100;
```

Cela aura pour effet de donner une priorité plus grandes aux routes fournies par le routeur RB, alors qu'elles sont en fait équivalentes à celles fournies par le routeur RC en termes d'étendue (pas forcément en termes de rapidité / bande passante)

Autre exemple d'application L'attribut LOCAL-PREF est souvent utilisé pour représenter les relations économiques entre les AS, elles sont définies de telle sorte qu'on ait :

$$rank(customerroutes) > rank(peerroutes) > rank(providerroutes)$$

Remarquons que cela permet l'utilisation de routes asymétriques pour l'aller et le retour, comme l'illustre le schéma suivant :



Dans cet exemple, les paquets devant aller de AS8 à AS1 ne passent pas par AS4 (c'est pourtant plus avantageux pour AS8 !) car AS4 n'a pas annoncé qu'il avait une route vers AS1 à AS8.

Filtres automatiques du protocole En plus des filtres définis par la configuration, les routeurs BGP appliquent les filtres suivans automatiquement :

- LOCAL-PREF est enlevé des routes avant qu'elles soient transmises
- L'ASN du routeur est ajoutée au début de l'AS-PATH des routes avant qu'elles soient transmises
- Une route contenant l'ASN d'un voisin ne sera pas transmise à ce voisin (sender-side loop detection)
- l'attribut NEXT-HOP est mis à jour avant que la route ne soit transmise

3.8 BGP interne (iBGP)

iBGP permet aux routeurs d'un AS de s'échanger des routes, il s'agit en fait d'une version modifiée de BGP dont les principales différences sont :

- LOCAL-PREF n'est transporté que sur les sessions iBGP (pas eBGP)
- les filtres d'import/export ne sont définis que pour les sessions eBGP
- les routes apprises dans les sessions iBGP ne peuvent pas être redistribuées sur d'autres sessions iBGP
- les attributs NEXT-HOP et AS-PATH ne sont mis à jour que lorsqu la route est envoyée à travers une session eBGP

Afin que tout se passe bien dans l'AS, il est préférable que tous les routeurs internes fassent tourner BGP (on appelle cela un full mesh).

Sinon, on peut :

- utilier des tunnels (MTU réduit, routeurs "chargés" par l'encapsulation/décapsulation des paquets aux extrémités du tunnel, configuration statique nécessaire (manuelle))
- redistribuer les routes BGP dans l'IGP (très mauvais pour la montée en charge, danger de réinjecter les routes dans BGP avec un AS-PATH de longueur unitaire (AS7007)

En résumé Le rôle d'IGP est de distribuer la topologie interne et les adresses internes à l'intérieur de l'AS, tandis que le rôle d'iBGP est de distribuer les routes vers les destinations extérieures apprises par eBGP aux autres routeurs de l'AS.

Les sessions iBGP peuvent être établies grâce aux route IGP.

Préférer eBGP à iBGP dans le processus de décision Aussi appelé "routage de la patate chaude", un routeur devrait tenter de se débarasser des paquets à destination d'autres domaines (AS) aussi rapidement que possible.

Choisir le next-hop le plus proche Toujours dans l'optique du "routage de la patate chaude", lorsque les règles précédentes du processus de décision amènent à des égalités, on utilise le next-hop (IGP, s'entend) le plus proche.

Choisir le MED le plus faible Alors que les deux règles qu'on vient de voir favorisent l'émetteur des paquets, le MED permet au futur récepteur d'influencer l'émetteur sur la route à choisir. Cependant l'émetteur peut choisir de simplement ignorer le MED. Celui qui annonce une route vers un certain préfixe peut ainsi joindre à la route un MED indiquant quelle route il préfère à une autre (plus le MED est faible, plus la route est favorable pour l'annonceur de la route)

Le MED impose toutefois un problème supplémentaire : deux MED ne peuvent être comparés que si les routes auxquelles ils appartiennent proviennent du même AS.

La règle "Choisir le MED la plus faible" dans le processus de décision va donc consister à garder, pour chaque AS ayant encore des routes "en lice", celle ayant le MED le plus faible.

3.9 Ingénierie de trafic basée sur BGP

3.10 Multi-homing

Au début de sa vie, un AS, connecté à un seul provider, utilise un ASN privé (de 64512 à 65535) et est complètement caché derrière son ISP.

Ensuite, l'AS veut devenir multi-homed et obtient un ASN public. l'ISP doit alors prévenir ses autres clients qu'un nouvel AS existe et faire ainsi grossir la taille des tables de routage globales.

Aggrégation BGP peut agréger les routes reçues, l'attribut AS-PATH est une AS-séquence formée d'AS-ensembles, la plupart des AS-ensembles sont constitués d'un seul élément, Un AS-ensemble n'est pas ordonné et compte pour une longueur de 1.

L'information sur le chemin réel est perdue lorsque les AS-ensembles sont utilisés ; mais l'aggrégation permet le diminution de la taille des tables de routage. Une liste des "mauvais élèves" n'utilisant pas l'aggrégation est maintenue. On estime que, si l'aggrégation était parfaitement utilisée, on pourrait réduire de 50% la taille des tables de routage.

Revenons à notre AS, il devient multi-homed Comme son ISP initial utilisait l'aggrégation, et que le nouvel ISP ne l'utilise pas (il ne peut pas, le préfixe du client n'entrant pas dans son propre préfixe), l'AS-PATH proposé par le nouvel ISP sera toujours utilisé (le préfixe est plus précis).

De son côté, l'ISP initial devrait arrêter d'agréger le préfixe du client, sinon le client ne recevra pas les paquets des autres clients agrégés (Le provider va annoncer une route contenant l'AS du client, le client va donc la rejeter, et n'aura pas connaissance des routes vers les autres clients du provider. Idem si les routes passent par d'autres routeurs : l'AS du client restera dans l'AS-PATH et la route sera donc systématiquement jetée (ou même pas reçue par le client avec le sender-side loop detection)

Comment limiter l'expansion des tables de routage ? Sur le long terme, IPv6 apportera une solution, en définissant une meilleure notion de multi-homing.

3.11 Gestion du trafic via BGP

On aimerait par exemple que le trafic vers un certain préfixe passe par un fournisseur, tandis que le trafic vers un autre préfixe passe par un autre fournisseur, ou alors privilégier un lien par rapport à un autre, mais garder l'autre comme back-up. Plusieurs techniques sont envisageables :

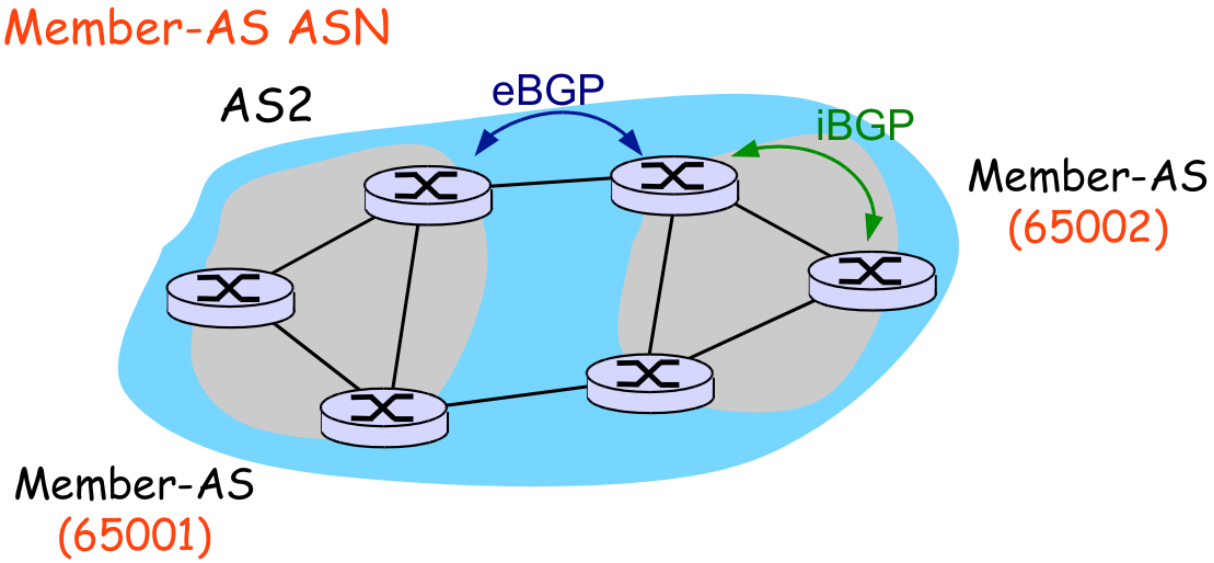
- Annonces sélectives : N'annoncer certain préfixes que sur certains liens (Expansion des tables de routage, que se passe-t-il si un lien lache ?)
- Préfixes plus spécifiques : Annoncer un préfixe global sur chaque lien, et un préfixe plus spécifique sur chaque lien (Expansion des tables de routage, mais plus robuste)
- AS-PATH prepending (rendre l'AS-PATH artificiellement plus long en ajoutant plusieurs fois son ASN à l'AS-PATH selon le lien sur lequel on se trouve à l'aide d'un filtre)

3.12 Résistance à la montée en charge

Objectifs

- Réduire le nombre de sessions iBGP (full mesh !), deux techniques pour cela : les confédérations et la réflection de routes
- Permettre une gestion des politiques de routage plus "scalable", la technique proposée est l'utilisation des communautés

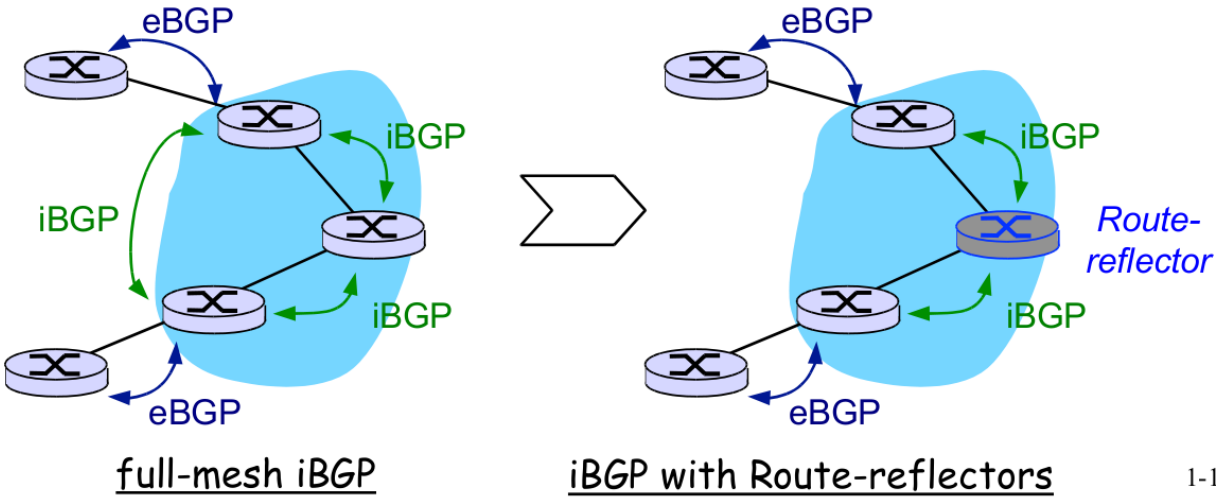
3.13 Confédérations



- iBGP est utilisé à l'intérieur de chaque AS-membre, eBGP est utilisé entre les AS-membres, chaque routeur a deux ASN : celui de la confédération et celui de l'AS-membre.
- Suivant la session eBGP, un routeur appartient à l'AS-membre ou à la confédération.
- Lors de la propagation d'un UPDATE via eBGP vers un routeur de la même confédération, un routeur insert son ASN-membre dans l'AS-PATH
- Il est autorisé d'annoncer un NEXT-HOP inchangé ainsi que l'attribut LOCAL-PREF.
- Lorsque qu'un UPDATE est propagé en dehors de la confédération via eBGP, le routeur doit d'abord enlever les ASN-membre et ajouter l'ASN de la confédération.

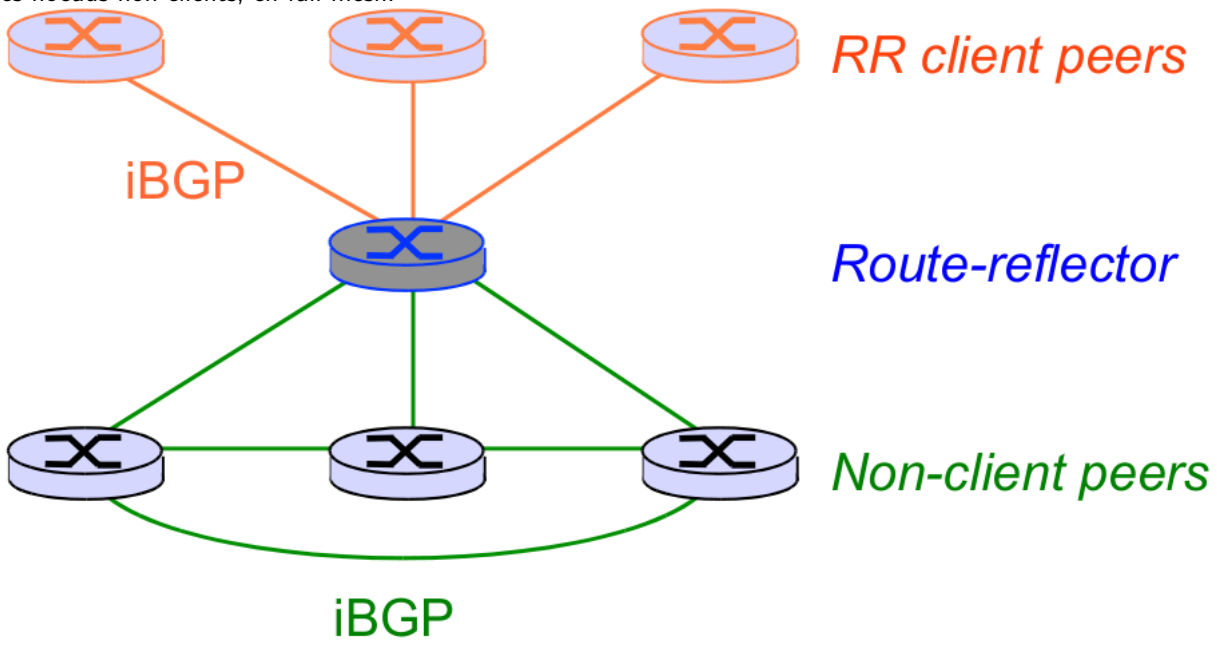
3.14 Réflection de routes

Des routeurs spéciaux appelés **réflecteurs de routes** peuvent propager sur une session iBGP des routes reçues d'une autre session iBGP.



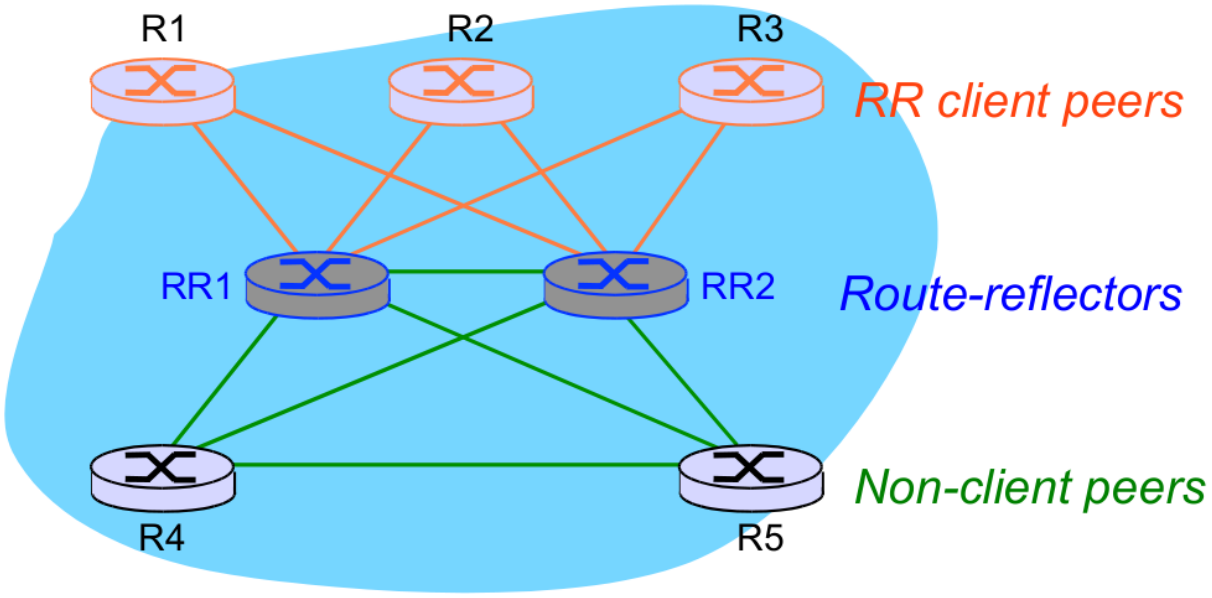
1-1

Il y a deux types de noeuds iBGP quand il y a des RR : les noeuds clients qui ne participent pas au full-mesh et les noeuds non-clients, en full-mesh.



Quand un RR reçoit une route d'un client ou de eBGP, il la redistribue à tout le monde.
Quand un RR reçoit une route d'un non-client, il ne la redistribue qu'aux clients.

Tolérance aux pannes Lorsque le RR tombe en panne, tous les clients sont déconnectés (single point of failure). Pour éviter cela, chaque client est connecté à au moins 2 RR.



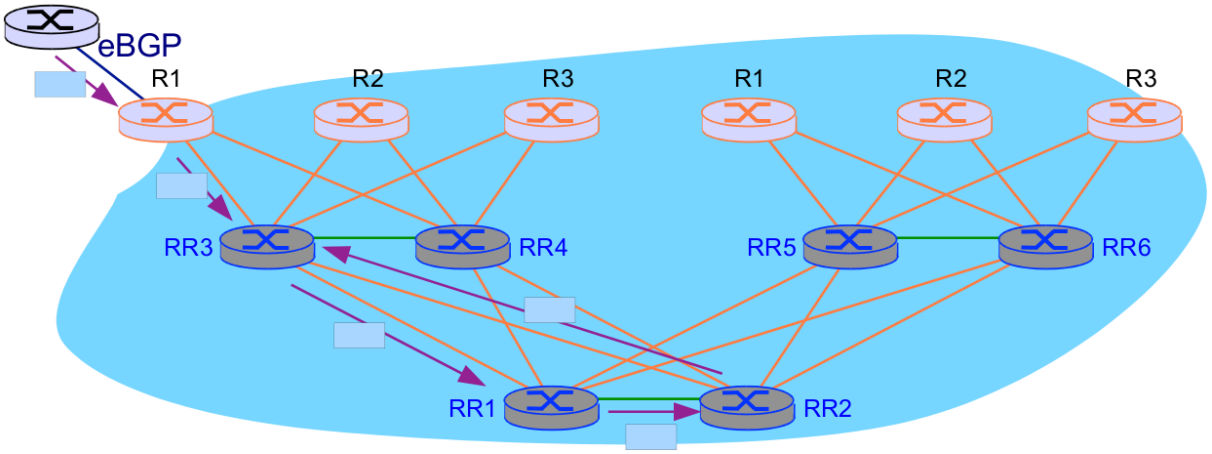
Problème : les boucles de routage sont possibles Nous devons être sûr qu'un update ne peut être retourné à celui qui l'a anoncé. Or, avec 2 RR, un client envoie l'UPDATE au premier RR, qui le transmet au second, qui le renvoie au client (boucle).

La solution à ce problème est l'attribut BGP ORIGINATOR-ID. Quand un RR réfléchit une route, il place l'identifiant du client qui a envoyé la route dans l'ORIGINATOR-ID.

Les routeurs se voyant dans l'ORIGINATOR-ID refusent systématiquement la route.

Hiérarchies de RR Un RR peut être lui-même client d'un autre RR, les mêmes règles de redistribution des routes s'appliquent.

S'assurer que les boucles de routage sont impossibles devient moins simple, ORIGINATOR-ID ne résoud plus ce problème.



Comme réponse à ce problème, on introduit un nouvel attribut : CLUSTER-ID-LIST ; quand un RR réfléchit une route, il ajoute son propre identifiant de cluster à la liste de la route.

Ensuite, pour éviter les boucles de routage, il suffit d'empêcher les routes d'atteindre un cluster par lequel elle est déjà passée.

Processus de décision, règles 9 et 10 On comprend à présent les tie-breaks 9 et 10 du processus de décision ; l'ORIGINATOR-ID et la CLUSTER-LIST proviennent en fait des clusters lorsqu'on utilise les réflecteurs de routes.

Comparaison entre les RR et les confédérations Avec les RR, les clients ne doivent pas être au courant de la hiérarchie, il suffit de configurer les RR, mais en contrepartie, la charge de travail est très forte sur les RR, qu'on installe en général sur les routeurs les plus puissants.

Avec les confédérations, il est simple de migrer d'une topologie multi-AS vers une simple confédération (par exemple, lorsqu'une société rachète une autre, elle peut vouloir intégrer le réseau de la société achetée au sien avec le moins de coûts possible). Le point noir des confédérations est que chaque routeur doit être configuré pour connaître l'id de sa confédération et de son AS-membre.

3.15 Filtres à grande échelle : les communautés

Une communauté est une valeur entière spéciale qu'on peut attacher à une route, elle est encodée sur 32 bits et deux routes avec la même communauté attachée sont généralement traitées de la même manière.

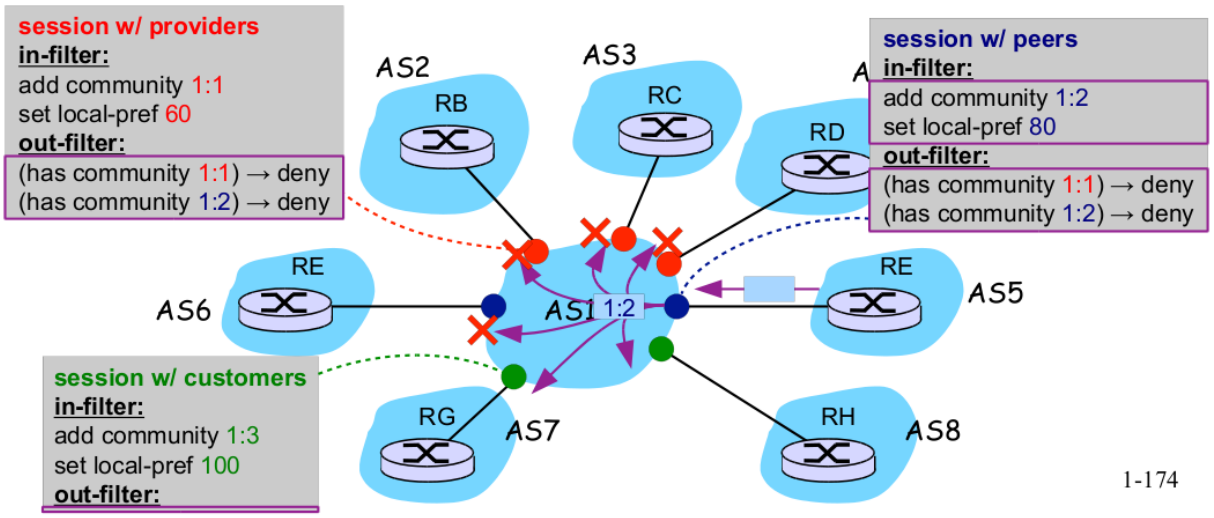
Certaines valeurs de communauté sont standardisées, par exemple $0xFFFFFF01$ (NO_EXPORT) et $0xFFFFFF02$ (NO_ADVERTISE).

Chaque AS dispose de 65536 valeurs de communautés pour lesquelles il peut définir la sémantique qu'il souhaite (ASN :0000 → ASN :FFFF)

La valeur d'attribut BGP COMMUNITY contient une ensemble de valeurs de communautés.

Exemple d'utilisation des communautés Souvent, les AS utilisent les communautés comme ceci : chaque routeur en relation avec un client applique un filtre à toute les routes reçues dont l'action est d'ajouter une communauté dont la valeur signifie "client", de même pour les routeurs en relation avec des pairs (valeur "pair") et les fournisseurs (valeur "fournisseur")

Ensuite, la politique peut être appliquée en fonction des communautés plutôt qu'en fonction des routeurs, ce qui simplifie généralement le travail des administrateurs réseaux.



Exemple plus complexe : ISP de recherche fournissant 2 types de services Les universités ont accès au réseau de recherche, les universités et les institutions gouvernementales ont accès à l'internet commercial.

Pour mettre cela en place, il suffit de "tagger" les routes (avec des communautés) apprises depuis des destinations de recherche ou commerciales. Ensuite il ne faut annoncer que les universités qu'au réseau de recherche et le réseau de recherche qu'aux universités.

ISP commercial fournissant 2 services de transit Le transit "plein" (annoncer toutes les routes connues aux clients, et les routes des clients aux autres clients, aux pairs et aux fournisseurs) et des routes clientes uniquement (n'annoncer à ces clients que les routes apprises par d'autres clients, n'annoncer les routes apprises par ce client qu'aux autres clients)

Autres utilisations On peut utiliser les communautés pour tagger la provenance des routes (pays), le point d'interconnexion d'où a été reçue la route, etc.

Une fois que les communautés sont apposées sur les routes, on peut les utiliser comme bon nous semble, par exemple faire des annonces sélectives sur bases des communautés, de l'AS-PATH prepending, etc.

Inconvénients Les communautés sont un attribut transitif. Dans certains cas c'est un atout, différents AS peuvent ainsi collaborer, mais en général ça donne surtout des routes polluées par un nombre parfois impressionnant de communautés, inutiles, mais les routeurs doivent quand même regarder ces communautés (consommation, mémoire, ...) et généralement les communautés ont une sémantique locale uniquement.

La meilleure chose à faire pour un routeur qui utilise les communautés, c'est de s'assurer que ces dernières ne sont pas annoncées inutilement à l'ensemble de l'internet.

3.16 Stabilité

3.17 Nombre de messages échangés

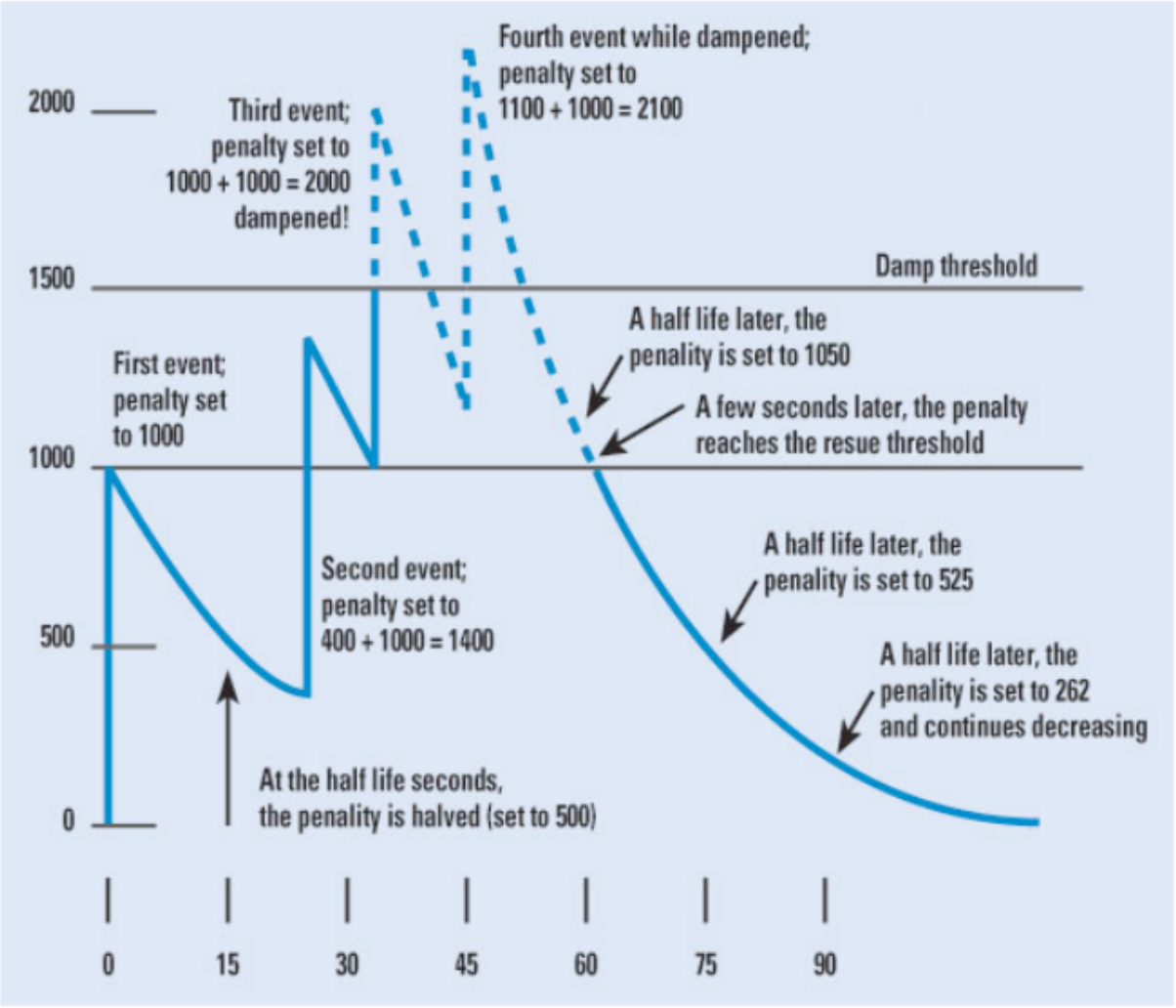
Bien que BGP soit un protocole incrémental, il n'en reste pas moins très bavard, pour plusieurs raisons :

- Les réseaux apparaissent et disparaissent
- Il y a toujours un routeur qui redémarre quelque part
- Disfonctionnements matériels, lignes arrachées, carte réseau qui "flappe", ...
- Mauvaise configuration
- Exploration du chemin (plusieurs itérations nécessaires avant de trouver la meilleure route)
- Bugs logiciels
- Instabilité IGP exportée à l'extérieur de l'AS (tie-break IGP et règle "NEXT-HOP le plus proche feront qu'un routeur pourrait changer de meilleure route sans arrêt)
- Algorithmes de routage secrets tentant des astuces suspectes ...
- Interactions politiques louches
- Gnomes, étincelles, fées, etc.

Comment réduire le nombre de messages UPDATE ? Le MRAI (Minimum Route Advertisement Interval) peut être augmenté (réduit le nombre de messages mais peut allonger le temps de convergence de BGP)

La plupart des routes ne changent pas fréquemment, une petite part des routes sont responsables de la plupart des messages BGP échangés. On peut associer un compteur pénalisant à chaque route, l'incrémenter chaque fois qu'une route change et utiliser un délai exponentiel pour lentement diminuer le compteur sur la durée.

Les routes avec un compteur trop élevé sont alors simplement supprimées pour un certain temps.



L'utilisation de cette technique (BGP dampening) est découragée aujourd'hui, car elle peut avoir un effet néfaste en diminuant la convergence pour des préfixes valides mais qui doivent faire une exploration du chemin avant de trouver leur meilleure valeur.

3.17.1 Stable Path Problem (SPP)

Raisonner quant à la conformité d'un système BGP est un problème toujours ouvert aujourd'hui.

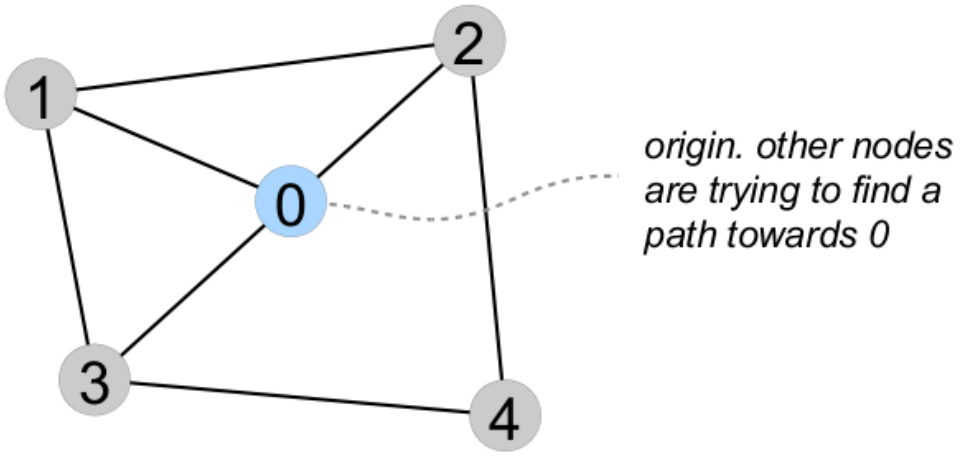
Le système a-t-il une solution ? Combien de temps durera la convergence ?

Le SPP est une première tentative pour formaliser la manière dont BGP fonctionne.

Un SPP est composé d'un graphe $G = (V, E)$, $V = \{0, 1, 2, \ldots\}$, $peers(u) = \{w | \{u, w\} \in E\}$, par convention, 0 est le noeud origine.

V={0, 1, 2, 3, 4}

E={(0,1), (0,2), (0,3), (1,2), (1,3), (2,4), (3,4)}



peers(1)={0, 2, 3}

Un chemin est une séquence $v_k, v_{k-1}, \ldots, v_0$de noeuds telle que chaque paire successive dans la séquence forme un arc dee V. Le chemin vide est désigné par ϵ. Chaque chemin non vide a une direction de son premier noeud v_k vers son dernier noeud v_0.

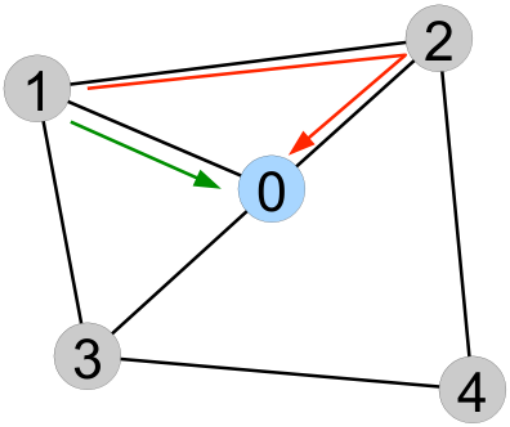
Pour chaque noeud v, P^v désigne l'ensemble des chemins permis de v à 0.

Si $P = (v, v_k, \ldots, v_0) \in P^v$, v_k est appelé NEXT-HOP du chemin P.

P^* est l'union de tous les P^v

P1=(1 2 0) and *P2*=(1 0) are paths from 1 to 0

P[1] could be limited to {*P1*, *P2*}, meaning that node 1 does not accept to go through node 3.



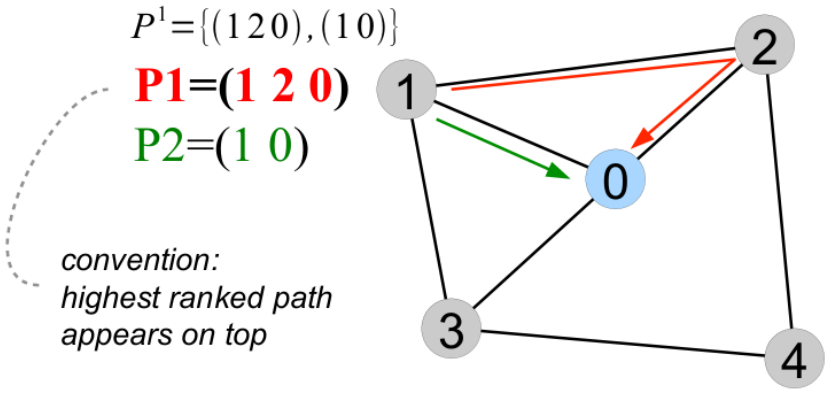
Note: (1 4 0) is not a path in G.

Le rang d'un chemin pour un noeud v est une fonction $\lambda^v : P^v \to R^+$ qui represente comment v range ses chemins permis.

Si $P1, P2, P^v$ et $\lambda^v(P1) < \lambda^v(P2)$, alors $P2$ est dit préféré à $P1$.

$\Lambda = \{\lambda^v | v \in V - \{0\}\}$

- $\lambda^1(P1) > \lambda^1(P2)$, or node 1 prefers path $P1$ over path $P2$



Une instance SPP $S = (G, P^*, \Lambda)$ admet $P^0 = \{(0)\}$.

$\pi(v)$ représente une assigation de chemin pour v. Si la valeur est ϵ, on considère qu'aucun chemin ne lie v à 0 dans l'assignation courante.

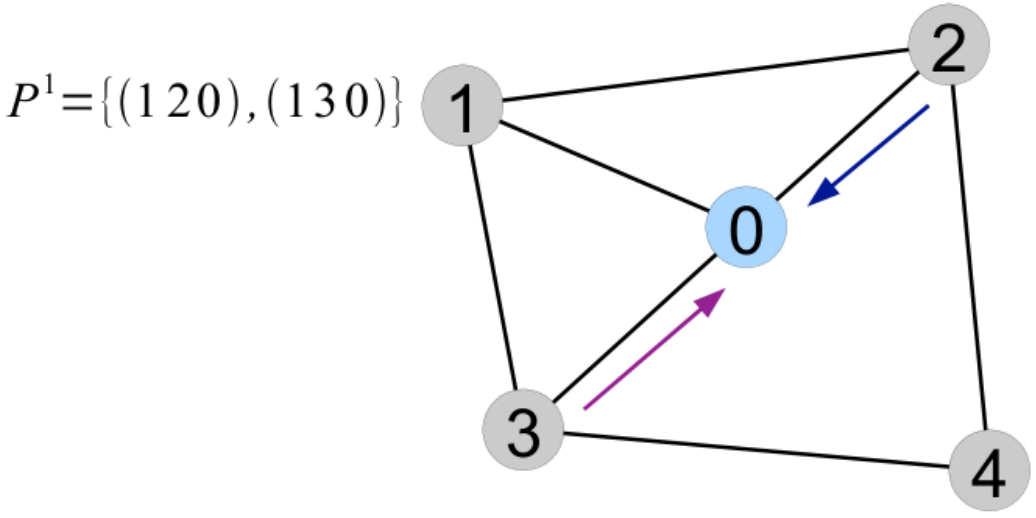
$choices(\pi, v)$ représente les chemins possibles, via l'assignation courante, que v a pour atteindre 0.

Le meilleur chemin P dans l'ensemble W pour un noeud u, noté $best(W, u)$ est tel que $\lambda^u(P)$ est plus grand que tout autre chemin de W.

Une assignation π est stable en u si $\pi(u) = best(choices(\pi, u), u)$ qui signifie que u est assigné au meilleur de ses choix.

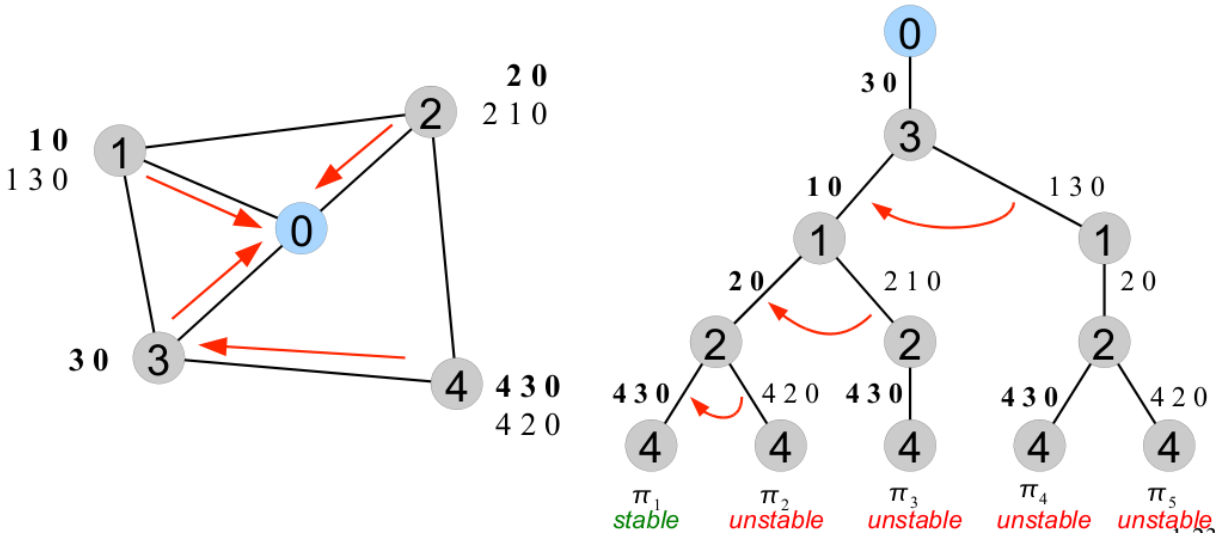
Une assignation est stable si elle est stable en chacun de ses noeuds.

- $\pi(1)=\varepsilon$; $\pi(2)=(2\ 0)$; $\pi(3)=(3\ 0)$; $\pi(4)=\varepsilon$
  $\rightarrow choices(\pi, 1)=\{(1\ 2\ 0), (1\ 3\ 0)\}$

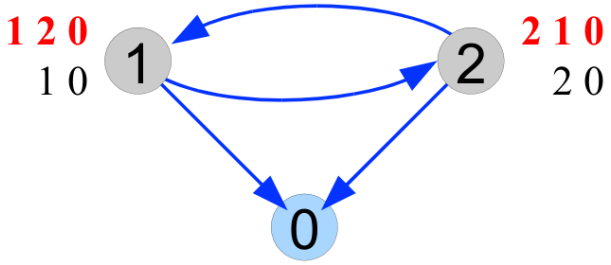


Une instance SPP est solvable si une assignation stable existe pour l'intance, on appelle cette assignation la solution de l'instance ; sinon l'instance est dite insolvable.

Une méthode pour trouver les assignements possibles, ainsi que les solutions, est de faire un tri topologique sur le graphe de dépendances, et ensuite d'ajouter les noeuds, dans cet ordre, jusqu'à ce que tous les noeuds soient ajoutés, on voit alors toutes les possibilités, pour choisir celles qui sont des solutions, il suffit de regarder qu'à chaque noeud de l'arbre ainsi construit, la branche est bien le meilleur choix.



Dans cet exemple, la solution est un shortest-path tree, ce n'est pas toujours le cas. Cette méthode se base sur le fait que le graphe de dépendances est un DAG, ce n'est pas une condition nécessaire pour que l'instance ait une solution. (Exemple : DISAGREE, dont le graphe de dépendances est ci-dessous)



Séquence d'activation Considérons maintenant le même modèle légèrement modifié, c'est-à-dire que chaque noeud est indépendant et voit une partie limitée du réseau. À chaque nouveau "temps", un noeud reçoit l'information d'un voisin, la traite et choisit une nouvelle meilleure route. L'ordre dans lequel les noeud s'activent est la séquence d'activation.

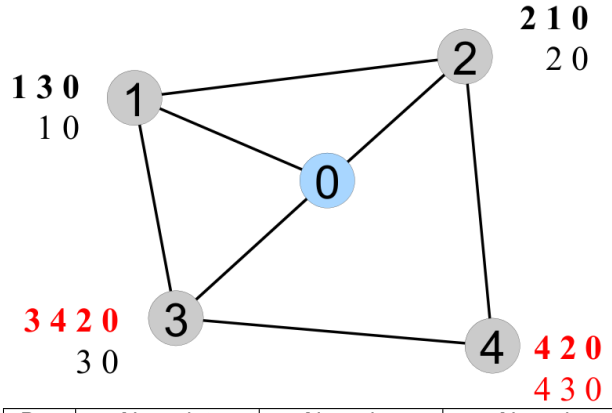
Exemple de séquence d'activation avec DISAGREE :

Step	Event	Noeud 1	Noeud 2
0	$(0,1) : \pi(0) = (0)$	choices={(1 0)} best=(1 0)	
1	$(1,2) : \pi(1) = (10)$		choices={(2 1 0)} best=(2 1 0)
2	$(0,2) : \pi(0) = (0)$		choices={(2 1 0), (2 0)} best=(2 1 0)
3	$(2,1) : \pi(2) = (210)$	choices={(1 0)} best=(1 0)	

On obtient dans ce cas une assignation stable, voici un autre exemple où les choses se passent moins bien :

0	$(0,1) : \pi(0) = (0)$	choices={(1 0)} best=(1 0)	
1	$(0,2) : \pi(0) = (0)$		choices={(2 0)} best=(2 0)
2	$(1,2) : \pi(1) = (1\ 0)$		choices={(2 0), (2 1 0)} best=(2 1 0)
3	$(2,1) : \pi(2) = (2\ 0)$	choices={(1 0), (1 2 0)} best=(1 2 0)	
4	$(2,1) : \pi(2) = (2\ 1\ 0)$	choices={(1 0)} best=(1 0)	
5	$(1,2) : \pi(1) = (1\ 2\ 0)$		choices={(2 0)} best=(2 0)
6	$(1,2) : \pi(1) = (1\ 0)$		choices={(2 0), (2 1 0)} best=(2 1 0)
7	$(2,1) : \pi(2) = (2\ 0)$	choices={(1 0), (1 2 0)} best=(1 2 0)	
...	...	...	...

Exemple d'instance de SPP insolvable Voici un exemple de séquence d'activation, on aboutira en fait toujours à un cycle (on considère ici que chaque noeud fait une action en même temps en reçoit les infos des autres noeuds en même temps).

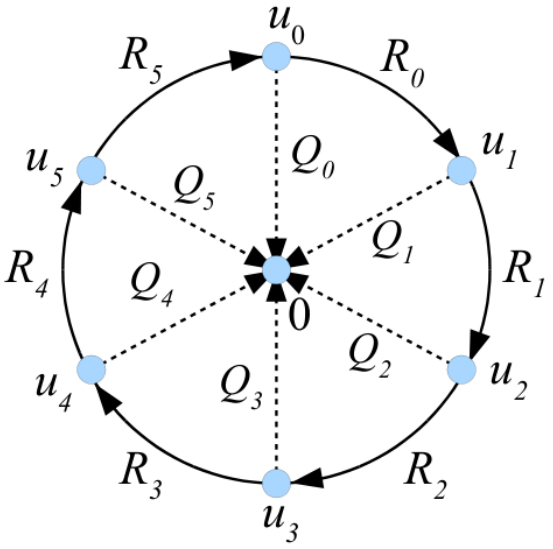


Pas	Noeud 1	Noeud 2	Noeud 3	Noeud 4
0	(1 0) (1 0)	(2 0) (2 0)	(3 0) (3 0)	
1	(1 0), (1 3 0) (1 3 0)	(2 0), (2 1 0) (2 1 0)		(4 2 0), (4 3 0) (4 2 0)
2		(2 0) (2 0)	(3 0), (3 4 2 0) (3 4 2 0)	(4 3 0) (4 3 0)
3	(1 0) (1 0)		(3 0) (3 0)	(4 2 0) (4 2 0)
4	(1 0), (1 3 0) (1 3 0)	(2 0), (2 1 0) (2 1 0)	(3 0), (3 4 2 0) (3 4 2 0)	(4 2 0), (4 3 0) (4 2 0)
5	(1 0) (1 0)	(2 0) (2 0)		$\emptyset$ ϵ
6		(2 0), (2 1 0) (2 1 0)	(3 0) (3 0)	(4 2 0) (4 2 0)
7	(1 0), (1 3 0) (1 3 0)		(3 0), (3 4 2 0) (3 4 2 0)	(4 3 0) (4 3 0)
8	(1 0) (1 0)	(2 0) (2 0)	(3 0) (3 0)	$\emptyset$ ϵ
...	...	...	...	...

Mauvaise nouvelle : la solvabilité est NP-complète, malgré ce résultat, des choses intéressantes peuvent malgré tout être dites.

Par exemple, ce n'est pas parce qu'un système est solvable qu'il convergera forcément. Une instance SPP est sûre si elle est solvable sous n'importe quelle séquence d'activation où aucune information n'est perdue.

Roues de dispute On peut voir DISAGREE comme une roue de dispute à deux rayons.



Q_i and R_i are non-empty paths

$\forall\, 0 \le i < k$, R_i path from u_i to $u_{(i+1) \bmod k}$

$Q_i \in P^{u_i}$

$R_i Q_{(i+1) \bmod k} \in P^{u_i}$

$\lambda^{u_i}(R_i Q_{(i+1) \bmod k}) \ge \lambda^{u_i}(Q_i)$

Q_i are *spoke* paths

R_i are *rim* paths

Théorèmes : Si une instance SPP n'a pas de roue de dispute, alors elle est solvable, a une solution unique et est sûre.

Limitations Le formalisme SPP est difficile à appliquer directement à un système BGP car les fonctions de rang ne sont pas explicites dans les filtres de routage BGP.

Le condition "Pas de roue de dispute" est trop forte dans certains cas.

Les politiques de routage ne sont toujours pas bien comprises et font l'objet de beaucoup de recherches.

Toutefois, des études ont prouvé que si chaque AS suit les deux lignes directrices suivantes, alors le système serait sûr :

- Ranger les routes de telle sorte que celles des clients soient préférées à celles des pairs, qui elles-mêmes doivent être préférées à celles des fournisseurs.
- Filtrer les routes pour que les chemins entre fournisseurs et/ou pairs ne soient pas autorisés.

3.17.2 Problèmes d'iBGP

- Les routes peuvent osciller à cause du MED
- BAD GADGET peut être "implémenté" en iBGP
- On peut créer des boucles de forwarding en iBGP

4 IPv6

4.1 Motivations

Les motivations principales sont :

- le manque d'adresses IPv4,
- le format des paquets IPv4 est complexe,
- le forwarding IPv4 est difficile à implémenter en hardware,
- des manques dans IPv4 comme de la configuration automatique, des qualités de services différentes, améliorer la sécurité et la mobilité, ... ,

Pour le moment on résout le manque d'adresses grâce aux **NAT** *(Network Address Translation)* ce qui permet en plus de cacher les adresses privées d'un réseau à l'internet. Cependant, traverser un **NAT** peut être compliqué, c'est également contre le principe de conversation "end-to-end" et ça donne un faux sentiment de sécurité. Il se peut également que les noeuds modifient le contenu d'un paquet (changer les adresses, les informations TCP/UDP,...).

4.2 Architecture d'adressage

Les adresses sont stockées sur 128 bits, ce qui fait 2^{128} adresses différentes soit $3,4 * 10^{38}$. Elles sont de 3 types :

- les adresses *unicast* : adresse identifiant une seule interface. Un paquet envoyé à une telle adresse est délivrée à l'interface identifiée par cette adresse. Dans ces adresses on compte :
  - $0:0:0:0:0:0:0:0$ (::), l'adresse non-spécifiée,
  - $0:0:0:0:0:0:0:1$ (::1), l'adresse de loop-back.
- les adresses *anycast* : adresse identifiant un pack d'interfaces (en général sur différents équipements). Un paquet envoyé à une telle adresse est remis à l'interface identifiée par cette adresse la *plus proche*. Ca fonctionne bien pour les transactions sans états (UDP par exemple).
- les adresses *multicast* : adresse identifiant un pack d'interfaces. Un paquet envoyé à une telle adresse est délibrée à toutes les interfaces identifiées par cette adresse. Par exemple, tous les pc's, GSM's ou autres doivent appartenir au groupe $FF02::1$, tous les routeurs à $FF02::2$, ...

De plus, les adresses seront fortement agrégeables, si une entreprise change de provider elle devra réaffecter des nouvelles adresses à ses interfaces. Aussi, si un sous-réseau est le "fils" de 2 réseaux différents, il aura une adresse pour chacun de ses 2 "pères" (il pourra alors être joint par l'un ou par l'autre avec l'adresse correspondante).

Il existe également un mécanisme permettant à 2 matériels proches, connectés sur le même lien ou LAN désireux de communiquer sans avoir accès à l'internet global. Chacune de ces adresses, appelées Link-local addresses, commence par FE8 (on peut par exemple les construire à partir de l'adresse MAC).

4.3 Paquets IPv6

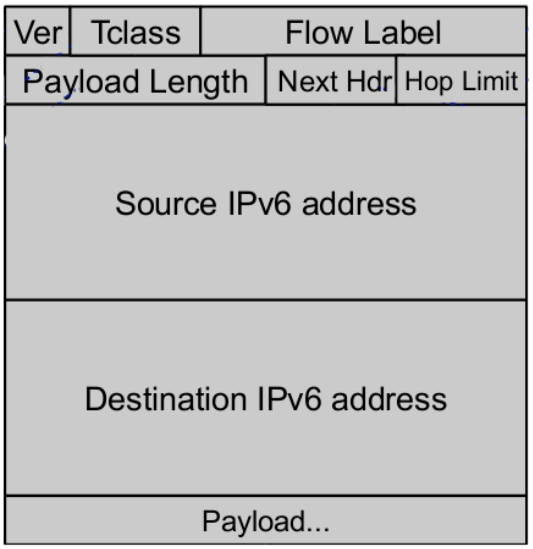


FIGURE 4 – Représentation d'un paquet IPv6

- Tous les champs sont alignés en blocs de 32 bits pour faciliter l'implémentation.
- Le TTL qui se renomme en Hop Limit.
- Next Hdr représente le type du payload (ex si $= 6$, le payload est un paquet TCP, il faut donc lire un header TCP, les headers étant chaînés les uns à la suite des autres).

IPv6 réclame un MTU minimum de 1280 bytes et la fragmentation n'est plus possible. Si elle est tout de même nécessaire, il faut fragmenter au niveau du cable, IPv6 ne s'en occupe pas (seuls les noeuds en fin de parcours ou en début de parcours font de la fragmentation/reconstitution avec le header d'extension de fragment).

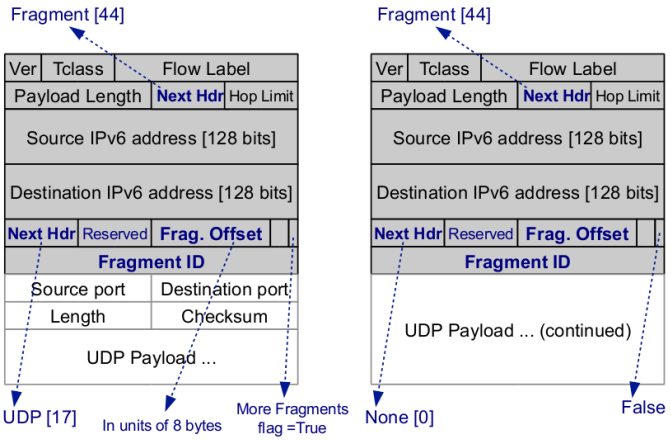


FIGURE 5 – Exemple de fragmentation en IPv6

4.4 ICMPv6

Il remplace l'**ARP** d'IPv4 en ce qui concerne la découverte de voisins. Il permet d'envoyer un message de sollicitation de voisin à l'adresse broadcast des noeuds sollicités ($FF02::1:FFxx:xxxx$) et de répondre par un message d'avertissement aux voisins qui ont envoyé une sollicitation (ou en multicast si la source n'était pas spécifiée).

Grâce aux messages ICMPv6 *(Internet Control Message Protocol)*, IPv6 est capable d'*auto-configuration* (**SLAAC**, *StateLess Address Auto-Configuration*). Cette auto-configuration utilise la découverte de voisins, la détection d'adresse dupliquée (DAD), la découverte de routeur ainsi que des préfixes et des paramètres des liens. Pour obtenir une adresse IPv6, un matériel peut avoir recours à différentes techniques. Soit comme en IPv4, manuellement codée par un opérateur ou via un serveur DHCPv6, soit via l'*auto-configuration* spécifiée ci-dessus. Celle-ci fonctionne comme suit :

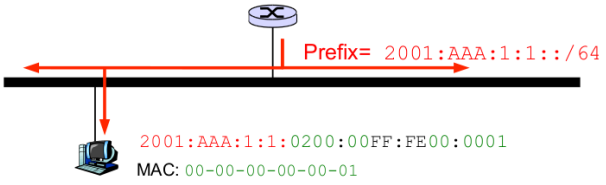


FIGURE 6 – Auto-configuration

- Le routeur envoie un préfixe en multicast ($FF02::1$),
- Les hôtes convertissent leur adresse MAC en format 64 bits (EUI-64) et la concatène au préfixe reçu du routeur

Lorsqu'un ordinateur démarre, il utilise son adresse link-local pour envoyer des messages ICMPv6 en vue de l'obtention d'une adresse IPv6. Il se peut cependant qu'un autre matériel possède la même adresse link-local (typiquement 2 matériels ayant la même adresse MAC ou des adresses configurées manuellement). Il doit donc détecter si quelqu'un utilise cette adresse, il utilise le DAD.

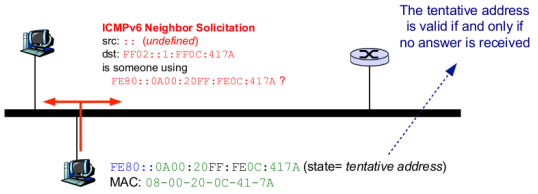


FIGURE 7 – Dupplicate Address Detection

Il envoie un message de sollicitation à l'adresse multicast des noeuds sollicités correspondant à l'adresse qu'il désire employer. (Le noeud doit souscrire à l'adresse multicast de tous les noeuds $(FF02::1)$ et à l'adresse multicast de tous les noeuds sollicités $(FF02::1:FFxx:xxxx)$ de l'adresse désirée. Une fois qu'il a son adresse link-local, il peut soit attendre que le routeur envoie son préfixe soit il peut lui envoyer un *router solicitation* afin que celui-ci lui envoie son préfixe. (Il faudra utiliser le DAD pour la nouvelle adresse créée également)

Le problème dans tout ça c'est que ces adresses auto-configurées utilisent les adresses MAC des composants, adresses qui sont fixes et uniques. Ainsi, n'importe qui peut être traqué très facilement $\Rightarrow$ problème de confidentialité ! On préfèrera donc l'emploi d'un serveur DHCPv6 configuré pour ne jamais réallouer la même adresse. On peut également autoriser les hôtes à utiliser des identifications d'hôtes aléatoires dans les 64 bits de poids les plus faibles de leur adresse IPv6.

Se pose maintenant le problème de la sécurité. En effet, il se peut qu'un utilisateur décide d'annoncer un faux neighbour solicitation ou un faux neighbour advertisement. Pour pallier à ça, la solution est le *CGA (Cryptographically Generated Addresses*. Chaque noeud possède sa clé privée et sa clé publique, chacun de ses noeuds aura pour adresse IPv6 le préfix du réseau + l'identifiant du subnet + la clé publique. Chaque noeud enverra donc ses données encryptées grâce à sa clé privée et le noeud la recevant utilisera la clé publique pour décrypter le message. Le problème c'est que les adresses IPv6 sont de taille fixe, et la partie disponible pour la clé publique est de 62 bits. Or une clé publique RSA de 62 bits ce n'est pas sécuritaire (au moins 1024 bits sont recommandés). L'idée est de prendre comme identifiant une fonction de hashage calculée sur la clé publique (en ignorants les 2 bits spéciaux pour la transformation en EUI-64). Ainsi, de manière simplifiée on a :

1. L'hôte A choisit une paire de clés $(Pub_A, Priv_A)$, et définit son identifiant d'hôte $HostId_A$ comme $Hash_{64}(Pub_A)$.
2. Un hôte B envoit une sollicitation de voisin pour l'adresse $prefix : HostId_A$ (prefix étant le préfixe du router).
3. A répond par un Neighbour Advertisement (NA) signé avec $s = E(H(NA), Priv_A)$ (E = encrypt). Il fournit également sa clé publique et son adresse de la couche lien.
4. B vérifie que $Hash_{64}(Pub_A) = HostID_A$ et que $D(s, Pub_A) = H(NA)$ (D = decrypt).

Le problème de **CGA** est qu'un hashage de 62 bits n'est pas vraiment sécurisant, en effet une attaque brute-force avec un dictionnaire peut avoir raison assez vite de ce système. L'idée pour résoudre ce problème est alors de calculer une plus grande valeur de hashage et d'en donner qu'une partie (la partie restante étant connue) et c'est ce qui est utilisé en pratique.

4.5 DNSv6

Chaque message DNS est composé de *Resource Record*'s (**RR**) encodés sous forme d'un quadruplet $(Name, Value, Type, TTL)$. Ces *RR* sont de plusieurs types :

- A, $Name$ représente alors un nom de domaine et $Value$ une adresse **IPv4**,
- $AAAA$, $Name$ représente alors un nom de domaine et $Value$ une adresse **IPv6**,
- NS, $Name$ représente alors un nom de domaine et $Value$ est le nom d'hôte d'un serveur **DNS** responsable pour $Name$,
- MX, identique pour les mails,
- $CNAME$, pour les alias.

Les DNS ont du être adaptés un tant soit peu pour supporter IPv6, ils ont par exemple été modifié pour supporter des réponses plus longues.

4.6 Transition d'IPv4 à IPv6

Il n'y aura pas de jour limite où tout le monde devra passer à IPv6, il faut donc qu'internet supporte les 2 versions. Pour celà il y a plusieurs mécanismes :

- La dual-stack, les noeuds suivant ce mécanisme possèdent un support pour les 2 protocoles. Ils agissent comme routeur IPv6/IPv4 lorsqu'ils communiquent avec d'autres noeuds IPv6/IPv4. Ceci impliquant un besoin de stockage plus grand car il faut tout stocker en double, comme une table de forwarding pour chacun par exemple.
- Le tunneling.

– La traduction qui consiste à employer des adresses IPv6 particulières dont la partie d'identification de l'hôte est une adresse IPv4. Un paquet IPv6 est alors transformer en paquet IPv4 par un NAT. Les soucis sont, entre autres, qu'il y a des champs du header IPv6 qui ne sont pas transformables en champs IPv4 et inversément, la fragmentation n'est pas supportée par IPv6, cela prend du temps et de la mémoire, il y a des contraintes sur la topologie (toutes les réponses doivent venir par le même NAT).

5 MPLS

La principale motivation de *MPLS (MultiProtocol Label Switching)* est de supporter de manière efficace des réseaux à grande vitesse c'est-à-dire rendre capable les routeurs de travailler sur un paquet et de le forwarder sur la bonne interface à la vitesse de la ligne. Il permet également d'effectuer du *Traffic Engineering*, il est utile pour les VPN's *(Virtual Private Network)* et est aussi très rapide au niveau de la restauration. Ce protocole utilise la technique du *Label Swapping*. Cette technique consiste à ce qu'un routeur analyse le label contenu par un paquet qu'il a reçu et l'interface d'où il est venu. Sur base de ces 2 informations (via une table de forwarding par label ($LFIB$)), le routeur décide sur quelle interface il doit pousser le paquet et quel label il doit lui accrocher.

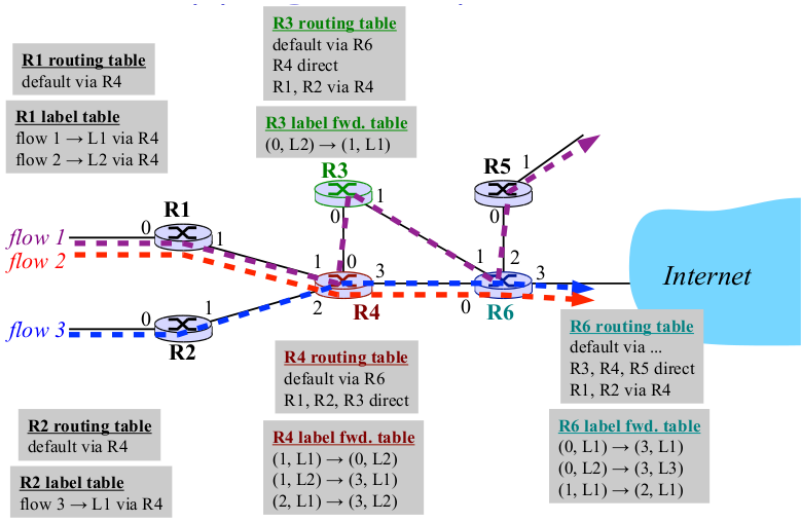


FIGURE 8 – Exemple de Label Swapping

5.1 Utiliser le label swapping et IP

L'objectif ici est d'utiliser le label swapping à l'intérieur d'un réseau et d'utiliser IP en dehors de celui-ci pour joindre d'autres réseaux. Pour construire un paquet IP contenant un label, il convient d'introduire un header spécifique de 32 bits en début de datagramme IP. Dans ce header, 20 bits sont réservés au label, ce qui offre $2^{20} = 1048576$ possibilités de labels différents. Un routeur pour alors effectuer 3 opérations :

1. **PUSH**, qui consiste à ajouter un label en début du datagramme IP,
2. **SWAP**, qui consiste à modifier le label en début du datagramme IP,
3. **POP**, qui consiste à enlever le label en début du datagramme IP.

Une *LFIB* est donc de la forme :

in-port	in-label	next-hop/out-port	opération
0	L1	2	POP

Un *LSR (Label-Switching Router)* peut gérer ses 2 labels de 2 façons différentes : soit par interface (et donc 2^{20} labels par interface) soit par LSR et donc 2^{20} labels en tout. Le chemin suivi par un paquet allant d'un ingress LSR (routeur d'entrée dans le réseau MPLS) et passant par quelques noeuds du réseau pour arriver à un egress LSR (routeur de sortie) est appelé *Label Switched Path* (**LSP**).

Comme on le voit sur la figure 9, les deux LSP's emprennent le même chemin de LSR6 à LSR5. Il est donc idiot d'implémenter 2 passages différents, on crée donc un petit LSP de LSR6 à LSR5 et les 2 autres LSP prendront ce petit LSP pour circuler dans le réseau.

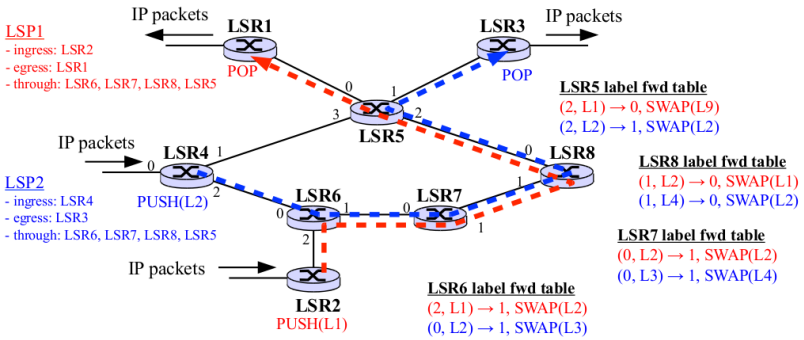


FIGURE 9 – Exemples de LSP

Comme on le voit, on a besoin de hiérarchiser les LSP. Le bit "stack" du header permet de spécifier que le label courant est au sommet de la pile ou pas.

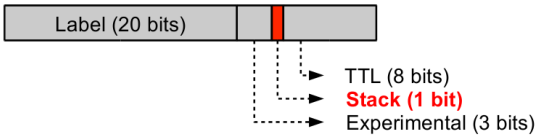


FIGURE 10 – Header MPLS

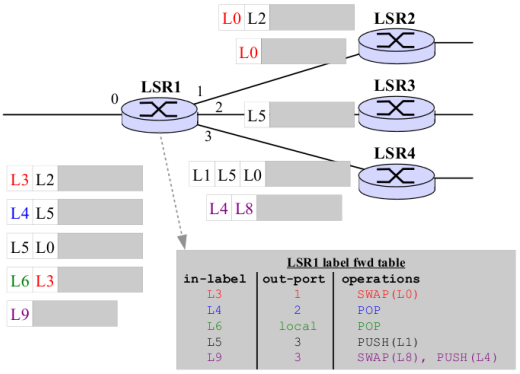


FIGURE 11 – LFIB

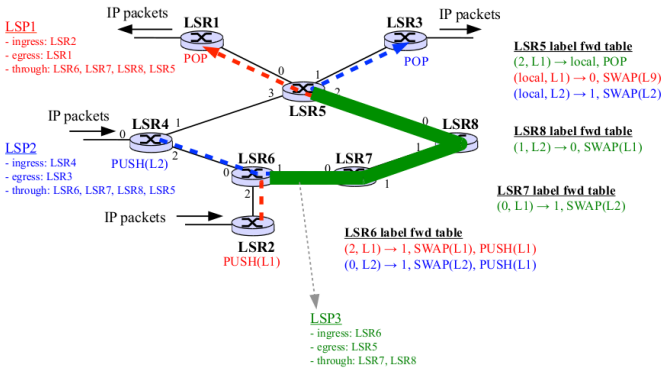


FIGURE 12 – Exemple avec stack

Les ingress LSR choisissent comment forwarder les paquets IP qu'ils recoivent. Ils définissent des **FEC** *(forwarding equivalence classes)* qui regroupent des paquets IP forwardés de la même façon. 2 paquets d'une même **FEC** auront donc le même label en sortant du ingress LSR. (On peut par exemple définir les **FEC** par destination, tous les paquets ayant même destination seront dans la même **FEC**)

5.2 Utilisations de MPLS

Forwarding de paquets basé sur la destination

La question est de savoir comment fournir un service de transit quand les LSR's de bordure sont capables d'attacher et enlever des labels, tous les LSR's supportent IP mais que les LSR's internes ne peuvent pas traiter les paquets IP de manière efficace ? L'idée est de créer un full mesh de LSP entre chaque paire de LSR de bordure. Il faut trouver dès lors un moyen de le faire automatiquement et d'enlever certains LSP inutiles.

On va utiliser un protocole spécial pour remplir les LFIB de tous les LSR d'un réseau, soit **LDP** *(Label Distribution Protocol* soit **RSVP-TE**. On pourrait également transporter des mappings FEC-label dans les messages de routage échangés par les routeurs mais cela implique que le protocole de routage doit être extensible (c'est le cas de **BGP** mais pas de **RIP**, **OSPF** ou encore **IS-IS** par exemple).

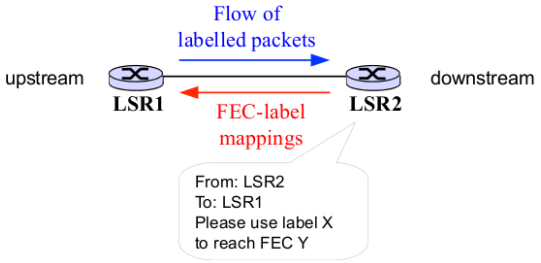


FIGURE 13 – Propagation des messages/mappings

Ces mappings sont envoyés de 2 façons différentes, soit sur demande soit de manière non-sollicitée.

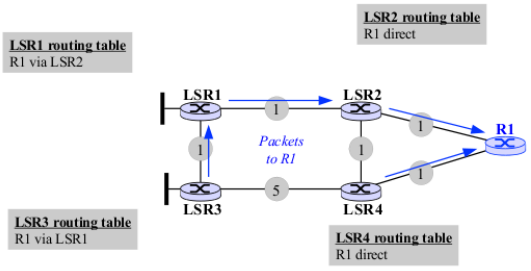
- Dans le premier cas, le LSR en début du flux envoie une requête de label et le LSR en fin de flux répond par un mapping FEC-label. Les avantages de cette méthode sont que les mappings ne sont envoyés que quand c'est nécessaire et les LSR ne doivent stocker uniquement que les mappings qui sont utilisés. Les inconvénients sont que lorsqu'un next-hop tombe en panne, du temps peut s'écouler avant qu'un label soit demandé au nouveau next-hop.
- Dans le second cas, ce sont les LSR en fin de flux qui envoient indépendamment leurs mappings au LSR de début de flux. Les avantages sont que chaque LSR peut recevoir plusieurs labels pour chaque FEC et dans le cas d'une panne passer d'un label à l'autre. Les inconvénients sont que les labels ne seront pas distribués au meilleur moment.

5.2.1 LDP

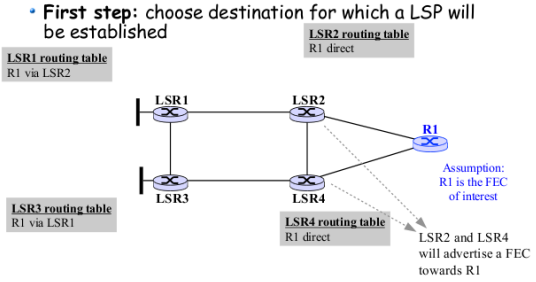
LDP utilise **UDP** pour faire de la découverte de voisins et **TCP** pour distribuer les mappings selon différents modes de transmission qu'il supporte. Il utilise plusieurs types de messages :

- Initialisation → établissement d'une session **LDP**,
- Keepalive → utilisé pour tester que la session est toujours établie,
- Label mapping → utilisé pour annoncer un mapping,
- Label withdrawal → utilisé pour enlever un mapping.

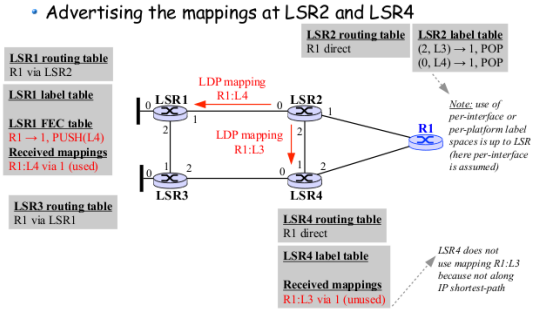
Périodiquement les LSR envoient à leurs voisins des messages LDP Hello à l'adresse multicast "tous les noeuds" (224.0.0.2), ces voisins répondent par un LDP Hello également s'ils sont LSR. LDP possède des qualités telles que la découverte automatique des voisins LDP, du transport de confiance, de l'extensibilité et le fait qu'il emploie les chemins les plus courts (ils se basent sur les coûts IGP). Voici un exemple d'établissement des liens LDP :



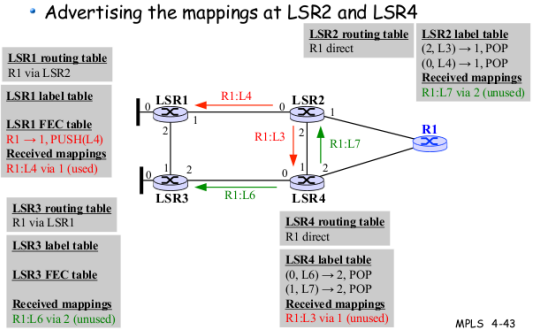
- **First step:** choose destination for which a LSP will be established



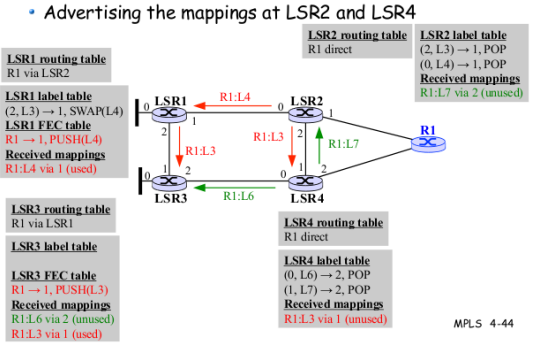
- Advertising the mappings at LSR2 and LSR4

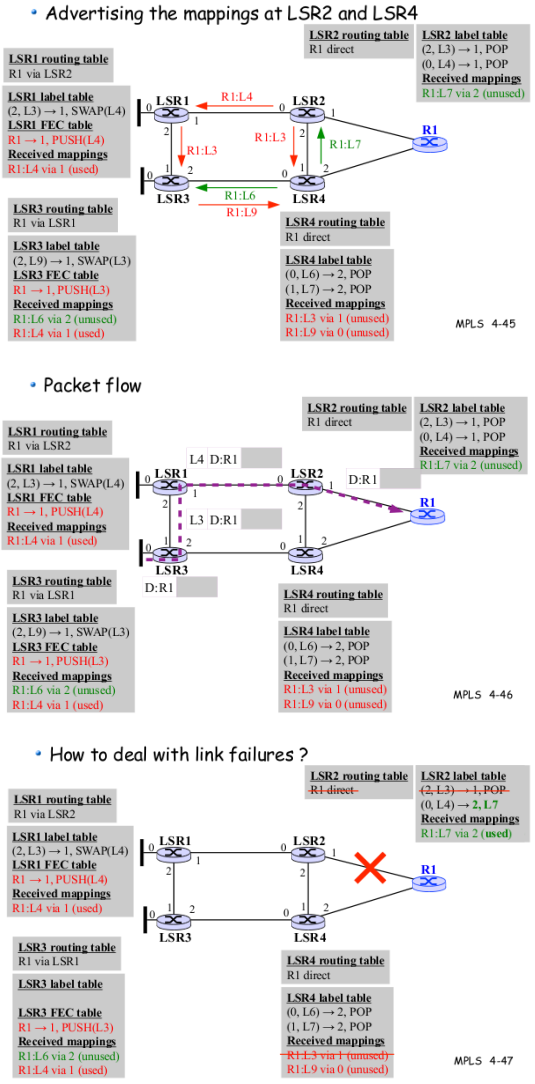


- Advertising the mappings at LSR2 and LSR4



- Advertising the mappings at LSR2 and LSR4





Traffic Engineering

L'idée est de développer un réseau IP normal ou IP+MPLS (pour que les paquets soient forwardés sur le plus court chemin). On collecte ensuite des statistiques aux routeurs en bordure sur la charge du trafic afin d'identifier les parties les plus congestionnées du réseau. Ensuite les ingress LSR établiront des sessions LSP le long de chemins bien choisis pour amener certains flux en dehors des zones congestionnées. Cette approche soulève 2 questions. La première est de savoir comment établir des sessions LSP avec des contraintes de qualité de service. Pour ce faire, on a besoin d'informations sur la capacité de chaque lien ainsi que d'un algorithme permettant de choisir le meilleur chemin en se basant sur ces contraintes.

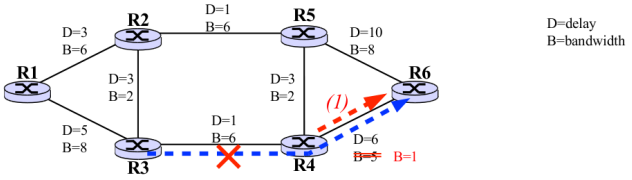
Pour les informations, il faudrait ajouter deux choses aux protocoles de routage déjà existants :

1. un moyen de distribuer l'information à propos de l'état courant ;
2. un moyen de calculer un chemin sujet à des contraintes.

Typiquement on va déployer celà sur des protocoles à état de lien tels que **OSPF** et **IS-IS**. En effet, il est difficile d'informer des routeurs de l'état des liens à distance dans des protocoles à vecteur de chemin/distance tels que **BGP** et **RIP**. De plus, avec les protocoles à état de lien :

- les routeurs coopèrent pour distribuer une carte du réseau → il devient simple d'ajouter de l'information à propos de la charge du réseau.
- les routeurs distribuent des paquets d'état de lien avec des informations de charge
  → le délai est déjà distribué comme la métrique IGP,
  → la bande passante/charge du lien est l'information principale à distribuer,
  → compromis entre la distribution fréquente et rare pour éviter la surcharge du réseau.

Le problème potentiel est le fait que les informations ne sont pas échangées immédiatement. Il se peut donc qu'un flot soit ajouté (ou du moins que l'on essaie de l'ajouter) sur un lien qui n'a plus assez de bande passante parce que le noeud n'a pas encore reçu l'information comme quoi il n'y avait plus assez de bande passante (cf Figure ci-dessous).



(1) a flow with B=4 is added between R4 and R6

(2) a tentative to add a flow with B=3 between R3 and R6 ***fails*** *because information about the change in capacity of link R4 → R6 was not yet received*

FIGURE 14 – Problème potentiel

Comment exprimer une contrainte ? Il y a 3 manières :
- **additive constraint** : trouver le(s) chemin(s) qui minimise(nt) $\sum d_i$ (comme pour le nombre de hop ou des délais ou couts des liens par exemple).
- **multiplicative constraint** : trouver le(s) chemin(s) qui minimise(nt) $\prod d_i$ (comme pour le taux de perte par exemple)
- **concave constraint** : trouver le(s) chemin(s) le(s) plus court(s) qui contiennent des liens qui respectent tous une contrainte donnée. (comme pour la bande passante par exemple)

Avec une seule contrainte additive ou multiplicative on utilise simplement Dijkstra, mais dès qu'il y a 2 ou plus de contraintes, le problème est NP-difficile. Heureusement, les contraintes concaves sont faciles à prendre en compte, il suffit simplement d'enlever les liens ne respectant pas la contrainte des choix possibles puis appliquer Dijkstra.

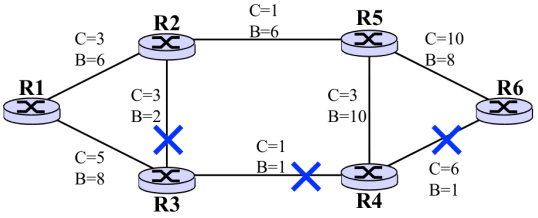


FIGURE 15 – Application d'une contrainte concave

Par exemple ici on cherche à trouver le chemin le plus court ayant au moins 3Mbps de bande passante, on enlève donc les liens (croix bleue) qui ne peuvent procurer cette bande passante.

Plusieurs chercheurs ont proposés des solutions pour le routage avec contraintes et aujourd'hui on a tiré les leçons de ces propositions :
- le routage avec contraintes doit être appliqué aux flux et non aux paquets un à un,
- la bande passante et le délai sont des contraintes clés (le délai jitter est moins important et plus difficile à prendre en charge efficacement)
- la sélection du chemin doit être fait par la source, aucun noeud du chemin ne prend de décision de routage. Si le flux envoyé sur le chemin n'est pas acceptable, la **source** devra en trouver(calculer) un autre.

Il existe des protocoles de routage avec contraintes, les plus connus sont $OSPF-TE$, $ISIS-TE$ et $PNNI$ (ATM).

Dans le cas de $OSPF-TE$, par exemple, les informations supplémentaires suivantes sont distribuées :
- le type de lien et son identifiant,
- les adresses IP locale et distante,
- la métrique de traffic engineering, qui est une métrique supplémentaire pour calculer les coûts des liens,

– la bande passante maximale,
– la bande passante réservable maximale,
– la bande passante non-réservée,
– classe/couleur de la ressource (peut être utilisée pour définir le type du lien).

5.2.2 RSVP

Les objectifs de ce protocole sont de supporter l'établissement de flux unidirectionnel dans les réseaux IP. (il est approprié pour l'unicast et multicast IP). Ce protocole utilise principalement 2 messages importants :

1. **PATH** message envoyé par l'expéditeur aux routeurs et récepteurs pour informer du nouveau flux et des ressources qu'il requiert MAIS aucune ressource n'est réservée via ce message. Un état de chemin est tout de même garder dans les routeurs jalonnant le chemin, mais rien de plus.
2. **RESV** message envoyé par les routeurs/récepteurs pour réserver les ressources pour le flux indiqué par le PATH. Ces messages remontent le chemin employé par le PATH, chemin qu'emploieront les datagrammes IP.

Un paquet RSVP contient plusieurs objets, identifiés par un champ "Class-num" dans le header du même objet. Voici les quelques objets connus :
– SESSION contient l'adresse IP de destination, le protocole IP et le port de destination,
– SENDER_TEMPLATE utilisé pour identifier l'expéditeur via son adresse IP et optionnellement le port,
– RSVP_HOP contient le hop précédent et le next-hop,
– SENDER_TSPEC définit les caractéristiques de traffic du flux.

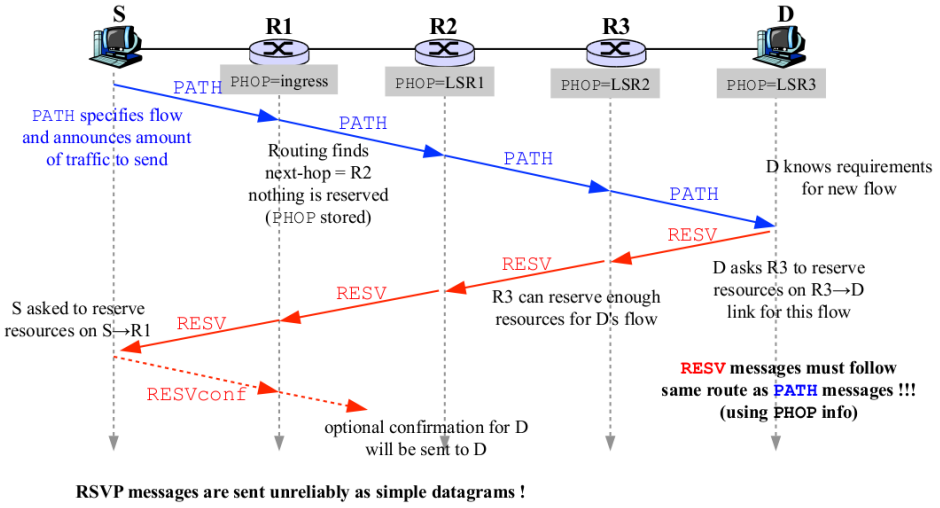


FIGURE 16 – Un exemple de RSVP

Les routeurs RSVP tiennent en mémoire un état par flux. Il est possible de gérer cet état de 2 façons :

1. de manière hard state, c'est-à-dire que l'état est créé à l'établissement du flux et supprimé quand ce flux est enlevé. Si un routeur intermédiaire crash, les réservations et l'état sont perdus. Cette solution est employée dans la plupart dans les réseaux "circuit-switched".
2. de manière soft state, c'est-à-dire qu'à l'état est associé un timer et quand celui-ci tombe à 0, l'état est supprimé. Les hôtes doivent alors s'échanger des PATH/RESV de manière périodique et si un routeur intermédiaire crash ou que le chemin change, l'état est automatiquement retiré.

RSVP utilise le soft state.

5.2.3 RSVP-TE

Il s'agit de l'extension de RSVP pour MPLS. Les ingress LSR envoient des messages PATH *(qui incluent un Label Request Object)* vers les egress LSR. Ceux-ci répondent par des messages RESV qui va propager le label à employer de hop en hop.

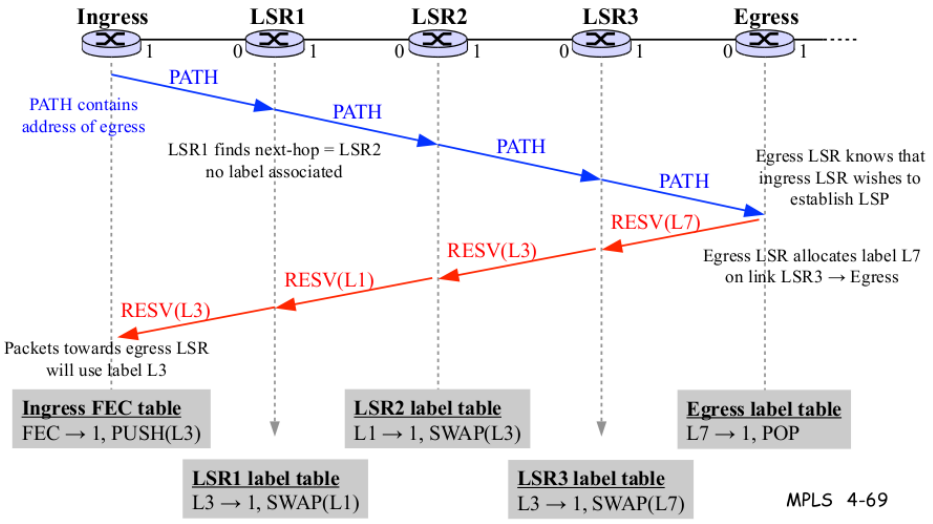


FIGURE 17 – Exemple d'établissement de session LSP avec RSVP-TE

Il est également possible d'établir ces sessions sur d'autres liens que les plus court. L'ingress LSR peut spécifier la route à emprunter pour établir le LSP via un **ERO***(Explicit Route Object)* (dans le message PATH). Cet objet contient une liste d'adresses IP, de préfixes de sous-réseau et de numéro d'AS. Il y a 2 types de routes :

1. Strict route, route dont chaque LSR spécifié **doit** être visité et uniquement ceux-là.
2. Loose route, route dont chaque LSR spécifié **doit** être visité mais où le LSP peut passer par un autre LSR entre 2 LSR's spécifiés.

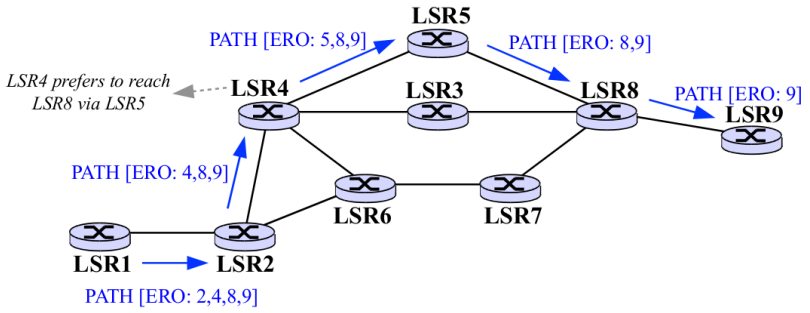


FIGURE 18 – Exemple d'emploi du ERO.

Problèmes à considérer

- Pour réserver la bande passante il faut utiliser le *Tspec* et le *Rspec* ⇒ Résolu.
- Pour supporter des flux de trafic variable, il doit être possible de modifier les ressources du LSP dynamiquement et s'il n'y a pas assez de ressources disponibles, que le LSP conserve les anciennes ressources.
- Comment changer dynamiquement la route employée ? (meilleur bande passante ailleurs ou un lien qui tombe dans la route actuelle)

6 Réseau de capteurs sans fil (Wireless Sensor Network - WSN)

6.1 Motivations

6.1.1 Tentative de définition

- Large quantité de noeuds-capteurs bon-marchés, peu gourmands en énergie et multi-fonctionnels déployés dans une région à intérêt.

– Capables de calculs
– Communiquent sur courte distance
– Collaborent entre eux pour accomplir des tâches communes

6.1.2 Exemple d'applications

réseaux intelligents, cités intelligentes, agriculture, construction, industrie, sécurité, santé, informatique portable, ...

6.1.3 Plus qu'un simple réseau sans fil

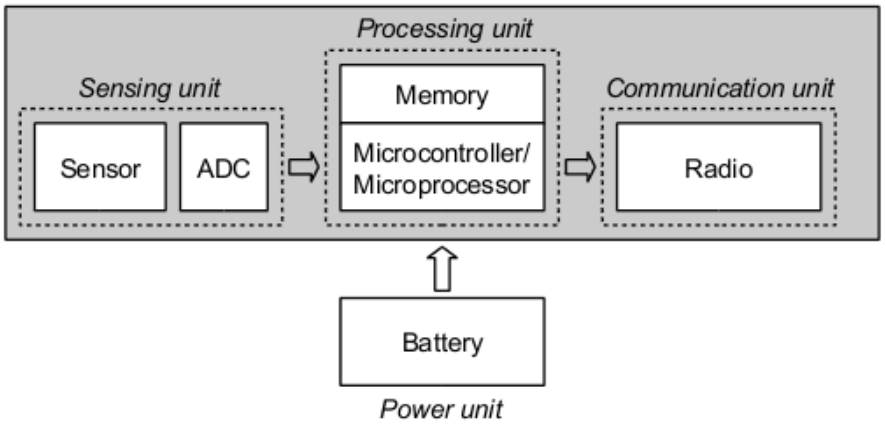
– Densité de noeuds plus élevée
– Contraintes plus grandes : énergie, stockage et calculs limités
– Auto-configurable
– Applications spécifiques : différent des applications d'un simple PC
– Manque de fiabilité plus élevé : changements de topologie fréquents
– Trafic de plusieurs noeuds vers un seul noeud (many-to-one traffic), de la source vers le puît.

6.1.4 Composition principale d'un WSN

– Miniaturisation du matériel
– Environnement de communication changeant : sans-fil avec perte et pas toujours allumé, besoin d'une nouvelle pile de protocoles réseau (routage, communication, ...)
– Application : temps réel requis, duty cycle, ressources limitées, informatique embarquée, tolérant aux défaillances du capteur
– Comportement adaptatif : condition de communication changeantes, environnement changeant, ...

6.2 Noeud-capteur

6.2.1 Structure typique d'un noeud capteur



6.2.2 Matériel typique d'un noeud capteur

– Processeur/microcontrolleur limité : lent (8 bits data path), basse fréquence (8MHz), pas d'opération à virgule flottante, pas de cache, pas d'unité de gestion mémoire, pas de MPU *(Memory Protecting Unit)*, pas d'instruction en pipeline.
– Mémoire limitée : 50KB de mémoire flash (pour le programme), 5-10 KB (pour la RAM)
– Radio de faible puissance : 50-100m de portée, non fiable, consomme quand même beaucoup d'énergie ($\sim 20mA$), débit de $100 - 200Kbps$
– Alimentation limitée : utilisation de batterie, énergie solaire, vibration des récoltes, ... (piles AA : $2500mAh$, soleil : $5mA/cm^3$)

6.2.3 Architecture d'un noeud capteur

- Exemple de radio : (voir slide 12)
- Exemple de capteur :
  - Passif : lumière, son ; humidité, pression, température ; vitesse angulaire/linéaire ; vibration, tension d'un matériel ; sensible à une substance chimique ; détecteur de fumée ; caméra ; ...
  - Actif : sonar, radar, sismique, ...
  - Actionneur : LED, relais, moteur, ...
- Exemple de source d'énergie
  - Énergie stockée : piles (non) rechargeables
  - Énergie captée : photovoltaïque, variation de température/pression, vibrations, flux d'air/de liquide
  - Métriques (en énergie par volume - J/cm^3) : piles rechargeables : $200mWh/cm^3$, soleil : $15mW/cm^3$, vibrations : $0.01 - 0.1mW/cm^3$

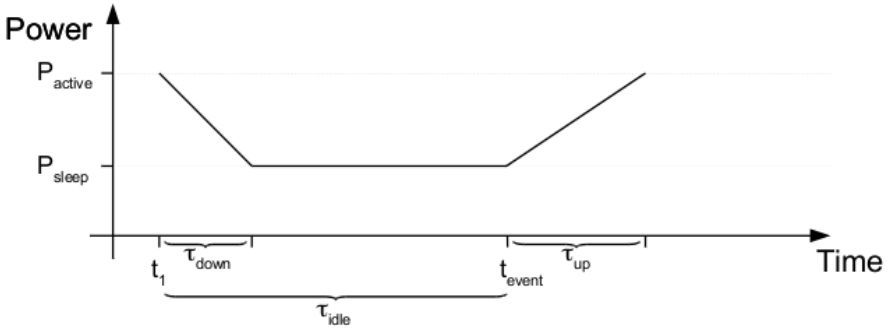
6.2.4 Consommation d'énergie - Microcontrôleur

Utilisation de techniques pour une consommation d'énergie efficace. Trois états possibles :

- actif : mode de fonctionnement normal, consommation max
- attente : fréquence du processeur faible, certains périphériques éteints
- repos : pas en fonctionnement mais fréquence basse, timer pour réveil

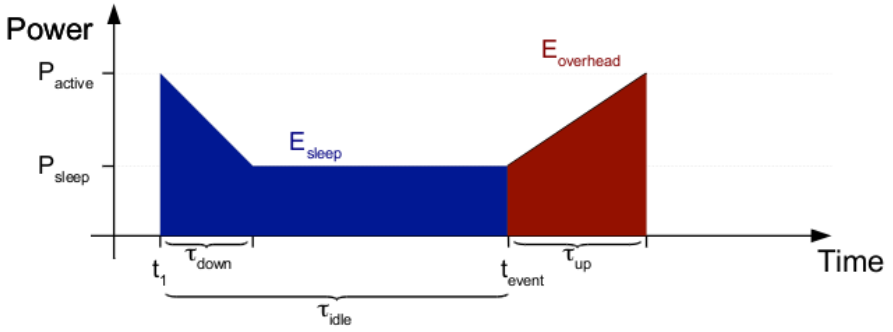
Mais les transitions entre les états ne sont pas libres car elles nécessitent de l'énergie. Plus l'état de repos est profond, plus il faudra d'énergie pour l'en sortir. Alors quand cela vaut-il la peine de changer d'état ?

Exemple : Supposons qu'au temps t_1, il n'y a rien à faire avant t_{event}. Faut-il aller dormir ou pas ? Aller dormir demande un temps τ_{down} et se réveiller τ_{up}. La consommation d'énergie vaut P_{active} si l'état est actif,

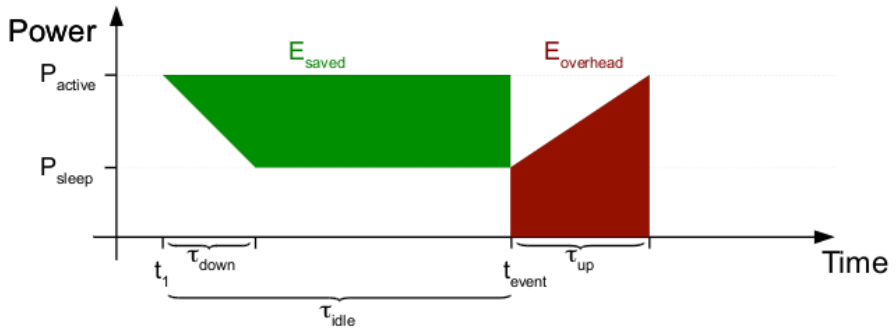


et P_{sleep} au repos.

- Si on reste actif : $E_{active} = \tau_{idle} \times P_{active}$
- Si on passe en mode repos : $E_{sleep} = \tau_{down} \times \frac{P_{active}+P_{sleep}}{2} + (\tau_{idle} - \tau_{down}) \times P_{sleep}$



$$
\begin{aligned}
E_{saved} &= E_{active} - E_{sleep} \\
&= \tau_{idle} \times P_{active} - (\tau_{down} \times \frac{P_{active}+P_{sleep}}{2} + (\tau_{idle} - \tau_{down}) \times P_{sleep}) \\
&= \tau_{idle} \times (P_{active} - P_{sleep}) - \tau_{down} \times \frac{P_{active}+P_{sleep}}{2} \\
E_{overhead} &= \tau_{up} \times \frac{P_{active}+P_{sleep}}{2}
\end{aligned}
$$



Partir au repos si $E_{overhead} < E_{saved}$ càd lorsque :

$$\begin{array}{rcl} \tau_{up} \times \frac{P_{active}+P_{sleep}}{2} & < & \tau_{idle} \times (P_{active} - P_{sleep}) - \tau_{down} \times \frac{P_{active}+P_{sleep}}{2} \\ & \Updownarrow & \\ t_{event} - t_1 & > & \frac{1}{2}(\tau_{down} + \frac{P_{active}+P_{sleep}}{P_{active}-P_{sleep}} \times \tau_{up}) \end{array}$$

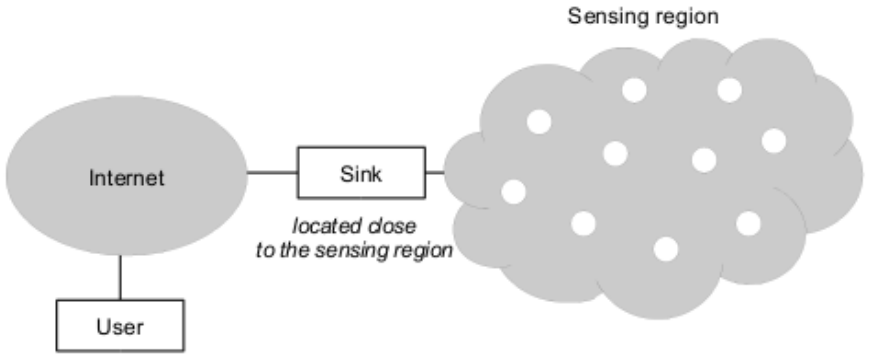
6.2.5 Consommation d'énergie - Radio

Une radio peut aussi se trouver dans différents états :
- transmission : émetteur-récepteur actif, antennes émettent de l'énergie
- réception : émetteur actif, réception en cours, amplificateur LNA consomme une grande partie de l'énergie
- attente : prêt à recevoir, pas de réception en cours, récepteur actif (LNA aussi), consommation d'énergie équivalente à l'état réception
- repos : plupart des émetteurs-récepteurs éteints (prend un certain temps à rallumer), difficile à uniquement allumer lors d'une réception

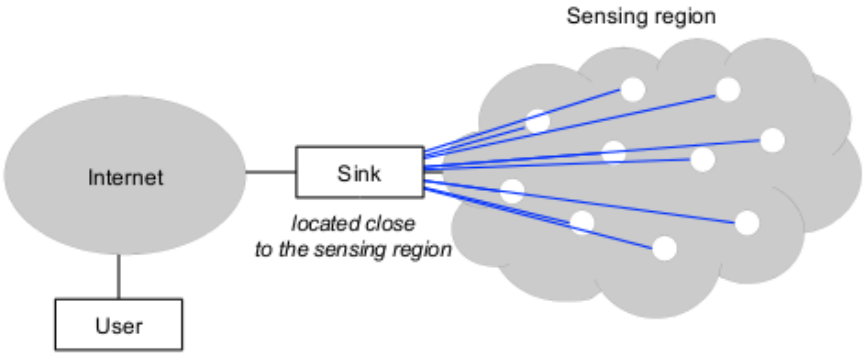
Il est possible d'adapter la puissance de l'émetteur, mais l'énergie consommée par l'émetteur n'est pas directement proportionnelle à celle produite par l'antenne (sous forme d'ondes). Donc réduire la puissance de l'émetteur ne permettra pas une forte diminution de l'énergie consommée.

Du côté du récepteur, peu de changements possibles. On peut éventuellement aller au repos quand c'est possible. Implique l'utilisation d'un cycle avec une période active pour écouter les autres noeuds et une période de repos lorsqu'on ne fait rien (*duty cycle*).

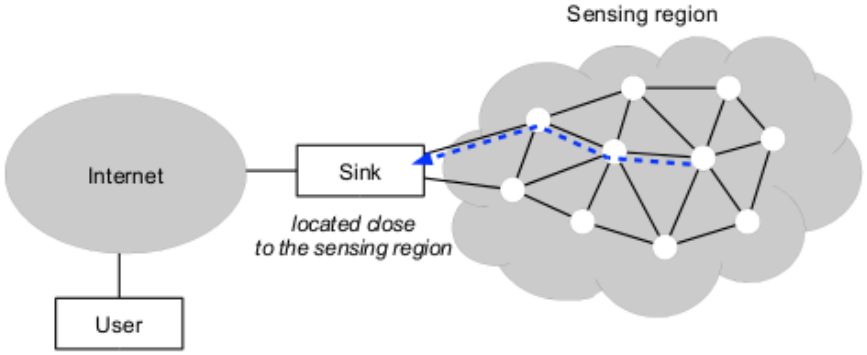
6.3 Architecture d'un réseau de capteurs



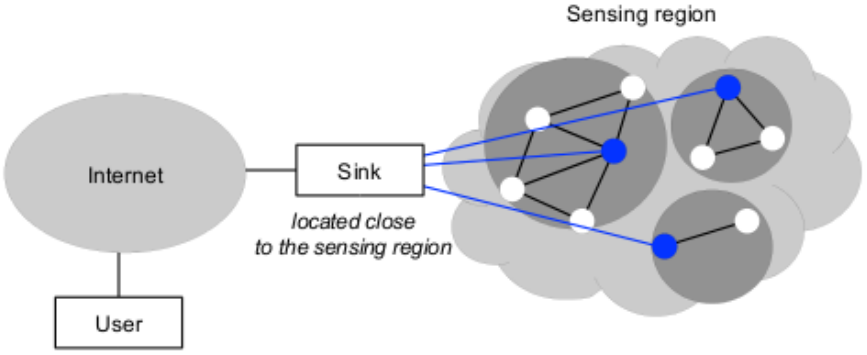
Différentes approches pour accéder à un noeud :
- single-hop : se fait sur longue distance, ce qui implique un haut coût de transmission (énergie augmente exponentiellement avec la distance).



– multi-hop : chaque noeud a le même rôle et peut forwarder une donnée d'un pair vers le puît sur des chemins multi-hop ⇒ diminution de la distance de transmission.



– clustering : ensemble de noeuds regroupés dans un cluster, des noeuds-puît rassemblent le trafic des clusters, le noeud principal d'un cluster envoit les données au puît principal ⇒ noeuds-puît doivent être plus puissants mais moins de perte d'énergie.



6.4 Couche MAC

6.4.1 Protocoles MAC traditionnels

– On ne peut pas utiliser CSMA/CD (collision detection) car cela suppose que l'émetteur puisse détecter les collisions → utilisation de CSMA/CA (collision avoidance).
– Les problèmes du terminal caché et des terminaux exposés sont règlés par l'utilisation de RTS/CTS mais ceux-ci introduisent un overhead significatif.
– Pourquoi ne pas utiliser des protocoles existants (Bluetooth, 802.11) ? Car le Bluetooth a besoin d'un maître permanent pour le polling et le 802.11 demande d'être constamment en écoute.

6.4.2 Contraintes demandées par le protocole MAC

– Doit conserver l'énergie : très différent des traditionnels WLAN.
– Extensible et robuste par rapport aux changements fréquents de topologie : noeud éteint temporairement, mobilité, déploiement de nouveaux noeuds, morts de noeuds existants.

Problèmes d'énergie

– Collisions : coûts de réception inutiles à la destination, coûts de transmission inutiles à la source. Éviter les collisions le plus possible.
– Overhearing : le réseau sans-fil s'appuie sur le broadcast, beaucoup de noeuds reçoivent des messages qui ne leur sont pas destinés et qu'ils droppent.
– Overhead du protocole : Contrôle de frame lié à MAC, header de paquets.
– Écoute en mode attente : consomme une énergie significative → aller au repos (duty cycle).

6.4.3 Différents protocoles MAC

– basés sur les conflits : CSMA protocols, S-MAC, Mediation device protocol
– basé sur un planning : LEACH, SMACS, TRAMA
– hybride : B-MAC, Z-MAC

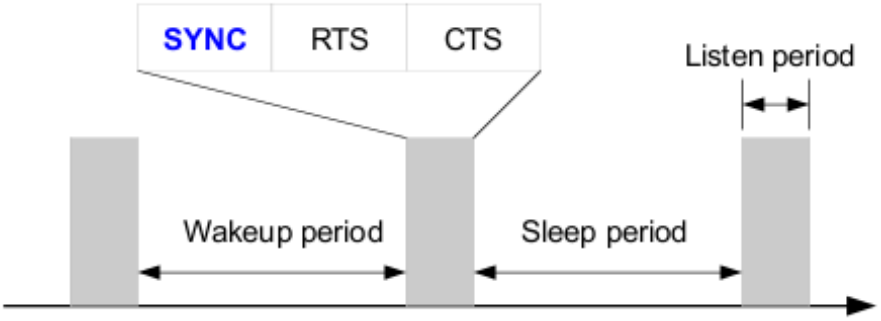
On étudiera ici les protocoles S-MAC et Mediation device ainsi qu'un cas d'étude : le IEEE 802.15.4.

6.4.4 S(ensor)-MAC

Objectifs :
– minimiser les conflits : utilisation d'un schéma CSMA/CA pour le broadcast et d'un schéma CSMA/CA avec RTS/CTS/DATA/ACK pour l'unicast.
– minimiser l'attente d'écoute : utilisation d'un duty cycle faible càd courte période d'écoute et longue période de repos.
– overhearing : les noeuds peuvent se reposer dès qu'ils entendent un RTS pour un autre noeud, chaque paquet indique la taille de transmission pour que les autres noeuds sachent quand se réveiller.

Schéma de réveil périodique : Un noeud alterne entre des périodes d'écoute et de repos selon son planning (schedule). La période d'écoute peut servir à la réception et la transmission. L'objectif de S-MAC est donc de tenter de coordonner le planning des différents voisins pour que leurs périodes d'écoute commencent au même moment.

Une période d'écoute est divisée en 3 phases : SYNC, RTS et CTS.

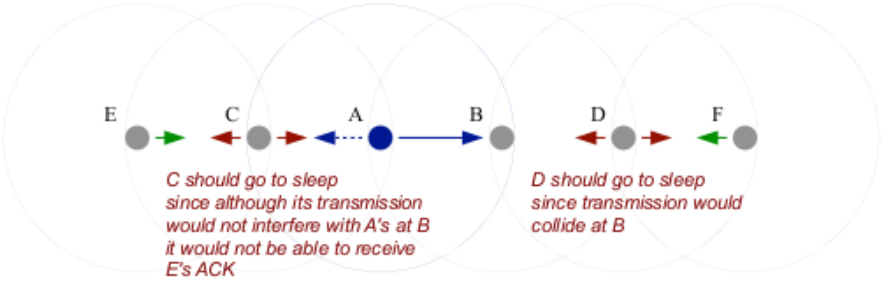


– SYNC : Le noeud accepte les paquets SYNC de ses voisins. Les voisins y notifient leur table de planning. Pendant la période de SYNC un schéma slotted CSMA/CA est utilisé (comme pour 802.11). Un paquet SYNC contient l'ID de l'envoyeur et le moment de son prochain repos.
  Un noeud qui veut transmettre durant la période de SYNC choisit un slot aléatoirement, vérifie si le canal est libre et commence à transmettre. Si le canal n'est pas libre, le noeud fait marche arrière jusqu'à la prochaine période de SYNC. Un noeud doit envoyer sa table de planning périodiquement mais pas à chaque réveil.
– RTS : Le noeud écoute les RTS de ses voisins (la canal est partagé en utilisant slotted-CSMA/CA comme pour SYNC).
– CTS : Le noeud transmet un CTS si un RTS a été reçu.

Synchronisation : Vouloir une synchronisation à l'échelle du réseau est difficile et peut augmenter la contention (tous les noeuds se réveillent en même temps et se disputent un nombre limité de slots). Solution : utilisation de clusters virtuels càd des groupes de noeuds qui partagent le même planning.

Au départ, un noeud qui s'allume écoute ses voisins. S'il reçoit un planning d'un voisin, il l'adopte comme son planning et en informe ses voisins par broadcast. Sinon, il choisit un planning aléatoire. Les noeuds qui ont le même planning appartiennent au même cluster. Si un noeud a choisit son propre planning, qu'aucun voisin n'a adopté son planning et qu'il reçoit un planning d'un voisin, il adopte ce nouveau planning. (exemple concret voir slides 45-50).

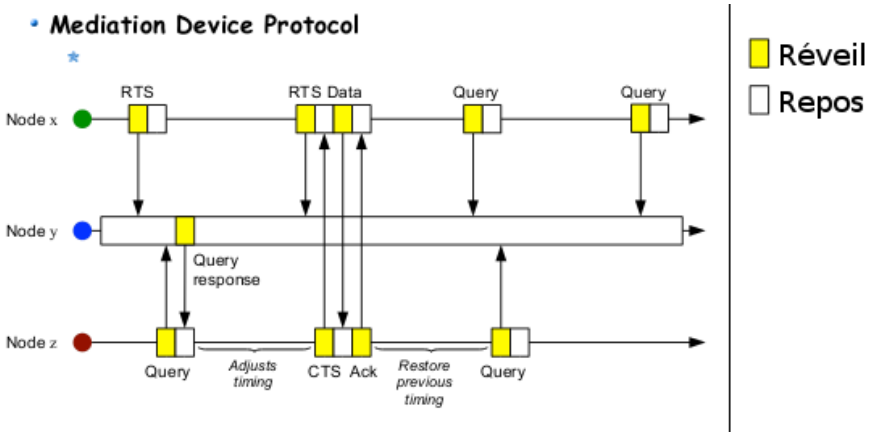
Limiter l'overhearing : Les noeuds vont au repos dès qu'ils reçoivent un RTS pour un autre noeud. Chaque paquet indique la taille de la transmission pour qu'un noeud sache quand se réveiller. Mais quels noeuds doivent aller au repos ? Les voisins du récepteur et de l'envoyeur.



Résumé S-MAC :
- Avantages : Les noeuds peuvent passer plus de temps au repos.
- Inconvénients : Utilisé avec un protocole de routage en maille, la latence peut être élevée car il faut attendre que les noeuds de la route se réveillent. La latence subie est d'à peu près $N \times T_{sleep}$ où N est le nombre de hops et T_{sleep} la période de repos.
- Améliorations possibles : Utilisation d'une écoute adaptative (adaptive-listening). Un noeud du next-hop peut plannifier une période supplémentaire d'écoute pour pouvoir écouter le RTS de son prédécesseur sur le chemin.

6.4.5 Mediation device protocol

Il n'y a pas de temps de référence global, chaque noeud peut avoir ses propres planning de repos. Le principe se présente comme suit :
- Quand un noeud se réveille, il transmet une courte requête "balise" (query beacon) indiquant qu'il est voulu de recevoir les paquets des autres.
- Ensuite, il reste éveillé pour un court moment, et si aucun paquet n'est reçu, il retourne au repos.
- Si un noeud veut transmettre un paquet à un voisin, il doit se synchronisé avec lui.
- Mediation device autorise les noeuds à se synchroniser sans devoir rester éveillé pendant un long moment.



Avantages :
- Pas besoin de synchronisation globale.
- La plupart de l'énergie est envoyée au dispositif de médiation.

Inconvénients :
- Nécessite un *mediation device* réactif à l'énergie
- Si plusieurs noeuds prennent le même planning, ils pourraient envoyer leur requête "balise" en même temps → collisions.

Travail supplémentaire :
- Réordonnancer les messages dans le cas de collisions répétées.
- Mediation device protocol distribué.

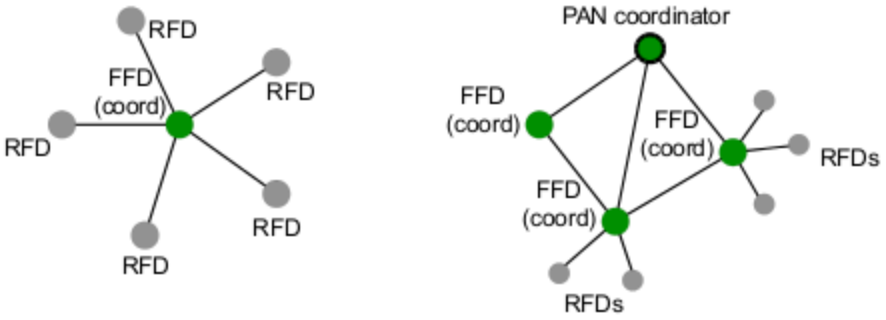
6.4.6 IEEE 802.15.4

Utilisation des couches physiques et MAC d'un réseau sans-fil personnel (WPAN) à bas débit.

Caractéristiques :
– Moyen à bas débits
– Dans les groupes industriels/scientifiques/médicaux
– Autorise les schémas basés sur la contention et la plannification

Architecture : Deux types de noeuds : les Full Function Device (FFD) qui peuvent travailler comme un coordinateur de réseau personnel (PAN), un simple coordinateur ou un device ; et les Reduced Function Device (RFD) qui sont de simples devices.

La topologie du réseau peut être en étoile pour connecter les devices au coordinateur ou en P2P pour connecter les coordinateurs entre-eux.



Adressage : Chaque noeud a une adresse unique codée sur 64 bits. Les premiers 24 bits représentent l'ID unique de l'organisation (OUI) alouée au fabricant par IEEE. Et les 40 bits suivants sont assignés par le fabricant.

Les noeuds peuvent aussi utiliser un adressage 16-bits pour diminuer l'overhead. Mais les adresses courtes ont une portée plus limitée et peuvent uniquement être utilisées dans un même PAN. Donc, les devices souhaitant atteindre un PAN extérieur le feront avec leur adresse courte + l'ID de destination du PAN externe (32 bits au total).

Format des frames :

Frame control	Sequ. number	Dest. PAN ID	Dest. Address	Source PAN ID	Source Address	Payload	FCS
2	1	0/2	0/2/8	0/2	0/2/8	variable	2

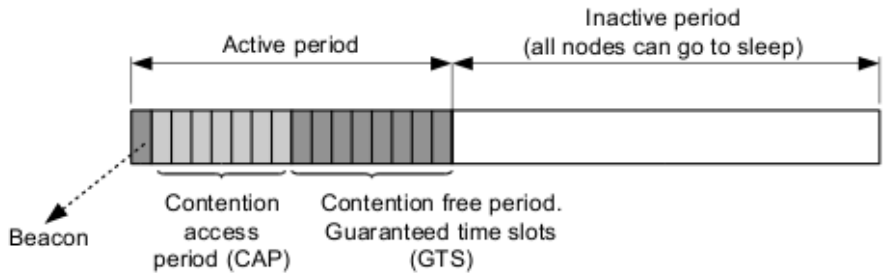
Frame type	Security enabled	Frame pending	Ack. request	Intra PAN	Reserved	Dest. address. mode	Reserved	Source address. mode
3	1	1	1	1	3	2	2	2

- Beacon (0)
- Data (1)
- Acknowledgment (2)
- MAC command (3)
- Reserved (4-7)

- No PAN ID / addr. Data (0)
  → only for Ack
- Reserved (1)
- Short address (2)
- Long address (3)

Rôle du coordinateur :
– Gérer la list des devices associés (association requête/réponse MAC command frame, utilisation des adresses 64-bits)
– Allouer une adresse courte aux devices
– Gérer la transmission des frames "balise" (beacon)
– Allouer un time slot garanti (GTS) en mode beaconed

Superframe : Le standard spécifie un mode *beaconed* où les canaux d'accès sont organisés par un coordinateur. Le beacon envoyé sur base régulière marque le début d'une superframe. La période active est divisée en 16 slots. Le premier est le beacon, le reste est réparti entre CAP *(Contention Access Period)* et GTS. Le coordinateur doit être actif durant toute la période active.



Accès au média durant CAP (contention possible) : Utilisation du protocole *slotted CSMA/CA* (synchronisation avec beacon). Chaque slot CAP est divisé en périodes de backoff.

Accès au média durant CFP (garanti sans contention) : Utilisation de TDMA (Time Division Multiple Access) (synchronisation avec beacon). Avant d'accéder au média durant le période CFP, un noeud doit faire une requête d'un time slot au coordinateur. La requête GTS contient le nombre de slots demandés, la direction (transmission ou réception) et indique s'il s'agit d'une requête d'allocation ou de désallocation. La fenêtre beacon contient la liste des adresses courtes qui sont autorisées à utiliser leur GTS pendant la période CFP de la superframe.

6.5 Couche réseau

6.5.1 Introduction

Objectif : La couche 3 fournit des chemins multi-hop. Beaucoup d'approches différentes sont possibles. On voit ici le protocole de routage MANET (Mobile Ad-hoc Network) :

- Divisé en protocole dirigé par des tables et des protocoles à la demande
- Optimized Link-State Routing (OLSR)
- Ad-hoc On-demand Distance Vector (AODV)

6.5.2 Ad-hoc On-demand Distance Vector (AODV)

Caractéristiques :

- Protocole de routage pour réseaux maillés.
- Fonctionne au dessus de IP, utilise UDP avec port 654
- Peut être utilisé sur des réseaux avec ou sans-fil
- 4 types de messages : Route Request (RREQ), Route Reply (RREP), Route Error (RERR) et Route-Reply Acknowledgment (RREP-ACK)
- Pas spécifique aux WSN
- Implémentation disponible dans TinyOS et Contiki

Fonctionnement : (slides 70-80)

Si un noeud veut communiquer avec un autre noeud hors d'atteinte :

- Le noeud envoie un RREQ en broadcast.
- Les noeuds qui le reçoivent mettent à jour leur table de routage avec une entrée pour le dernier envoyeur du message et le créateur du message.
- Ils broadcastent à leur tour le RREQ.
- Quand le noeud destination reçoit le RREQ, il met à jour sa table de routage et renvoie un RREP en unicast.

Détails supplémentaires

- Recherche en cercle : RREQ envoyé avec un TTL. Si aucune réponse, renvoie avec un TTL plus élevé.
- Mécanismes de flooding : Utilisé pour le broadcast des RREQ. Un noeud garde la trace du numéro de séquence du créateur du message dans le RREQ. Si un autre RREQ du même créateur est reçu, il est forwardé si le numéro de séquence est plus grand que l'ancien, il est jeté sinon.

- Limite du flooding : Un noeud qui reçoit un message RREQ et qui a déjà une entrée pour la destination dans sa table de routage et dont le numéro de séquence de l'entrée est $\geq$ au numéro de séquence du RREQ, alors le noeud répond avec un message RREP et ne forwarde pas le RREQ.
- Défaillance du lien : Des *Hello messages* sont envoyés régulièrement par un noeud sur une route active vers ses prédécesseurs. Si aucun *Hello* n'est reçu dans un certain laps de temps, la route est considérée comme invalide et une procédure de recovery doit être lancée.

6.6 Couche transport

TCP pour WSNs ? TCP a été conçu pour les réseaux filaires qui ont un faible taux de perte de paquets dû aux erreurs, et un haut taux de perte de paquets dû à la congestion.

- Perte de paquet interprétée comme congestion → réduction de la fenêtre de congestion → réduction du débit.
- Principe de end-to-end → considère que les noeuds intermédiaires sont muets et ne peuvent pas prendre part à une retransmission.

Problèmes avec TCP :

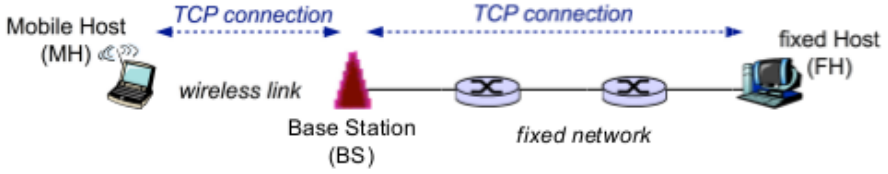
- Ne fonctionne pas bien avec les réseaux sans fil à cause du haut taux d'erreur (Bit Error Rate)
  - paquets perdus à cause d'erreur de bit : 5-10% au plus
  - paquets perdus impliquent une retransmission et réduction de la fenêtre de congestion malgré qu'il n'y ait pas de congestion
  - les paquets peuvent être retransmis par la couche lien mais cela augmente le RTT → augmente le RTO → TCP réagit lentement aux changements de congestion.
  - injustice augmentée : les plus longs chemins ont une plus grande probabibité d'erreur → dégradation des performances
  - ...
- Les retransmissions TCP ent-to-end sont nuisibles pour la contrainte énergétique des WSN car plusieurs noeuds pourraient avoir besoin de retransmettre sur un chemin multi-path.

Solution proposée :

- Amélioration de TCP avec des liens sans-fil : split connections (I-TCP, Split-TCP), solution de la couche lien (Snoop)
- Amélioration de TCP dans les WSNs : Distributed Caching Protocol (DCP)
- Autres protocoles de transport : pas vu ici

Split connection (I-TCP) :

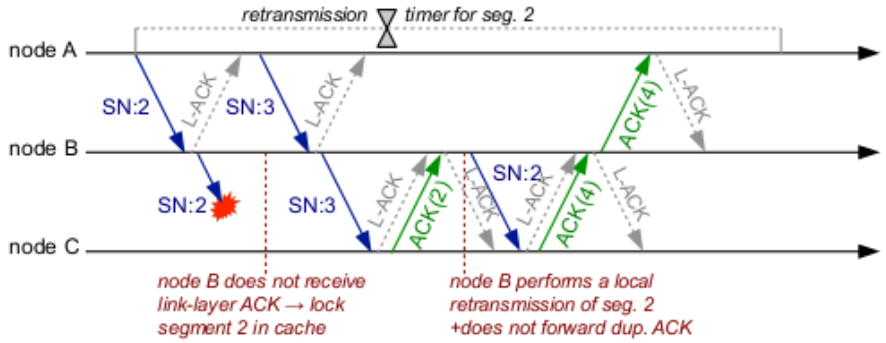
- Idée de base : La connection TCP est splittée à la station de base (BS), ensuite, deux connexions sont établies et la BS copie d'une connexion à l'autre.



- Avantages : repose exactement sur le même protocole TCP, petits RTT pour chaque connexion (recovery plus rapide), les pertes sans-fil sont cachées de la connexion entre BS et FH, et la connexion entre MH et BS peut utiliser un protocole de transport affiné.
- Inconvénients : Viole le principe *end-to-end* de TCP, de la mémoire et du CPU supplémentaire sont requis à la BS (buffer des 2 conn, controle de conn*2 et copie d'une conn à l'autre), et transfert plus complexe (nécessite un état à changer de la BS de base à la nouvelle BS).

Distributed Caching Protocole (DCP) :

- Idée de base : Implique les noeuds intermédiaires le long d'un chemin multi-hop. Essaye de réaliser des retransmissions locales autant que possible. Repose sur des connaissances positives de la couche lien pour sélectionner ce qui doit être mis en cache.
- Principe de fonctionnement : Chaque noeud doit être capable de mettre en cache quelques segments TCP. Met en cache les segments de haut num de séq. Vérrouille les segments en cache qui sont un-ack'd. soft-state dans les noeuds intermédiaires



– Avantages : la retransmission se passe plus près de la destination → celle-ci n'implique pas tous les noeuds du chemin multi-hop.
– Inconvénients : Les noeuds intermédiaires ont besoin de mémoire additionnelle pour mettre en cache les segments et les opérations supplémentaires. Difficulté de prédire les améliorations en présence de faible cache et d'une large fenêtre d'envoi (besoin de simulations/mesures expérimentales).

6.7 Modèle de programmation / RTOS (real-time operating system)

6.7.1 Objectifs

– Comprendre comment la programmation des systèmes embarqués en réseaux diffère de la programmation logicielle ordinaire (pas d'OS de support, programmation séquentielle vs événementielle).
– Découvrir comment une pile IP peut tenir dans un très petit système (uIP/lwIP).
– Avoir un contact avec les WSN récents orientés RTOS (TinyOS/nesC, Contiki/Protothreads)

6.7.2 Différence avec la programmation ordinaire

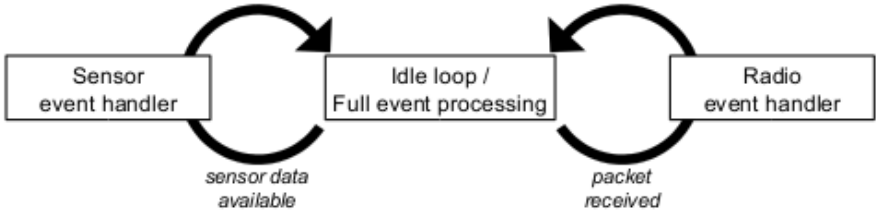
OS de support La programmation logicielle traditionnelle repose fortement sur les services apportés par l'OS :
– contrôle/protection des accès aux ressources
– gestion de l'allocation des ressources à différents utilisateurs
– support pour les exécutions concurrentielles de pls processus
– communication entre processus

Exigences des noeuds WSN :
– Les microcontrôleurs n'ont généralement pas assez de ressources pour accueillir un OS complet (pas de MMU *(Memory Management Unit)*, mémoire limitée, ...).
– Les exigences pour la concurrence sont différentes (unique user, multiples tâches).
– Le peu de mémoire nécéssite souvent des schémas d'allocations statiques ou dynamiques.

Programmation séquentielle Les applications traditionnelles sont souvent conçues autour d'un modèle de programmation séquentielle. Ce modèle pourrait mener à des pertes au niveau des capteurs de données ou de la radio.

Processus basé sur la concurrence : Un OS fait attention au partage CPU et fournit un environnement d'exécution en parallèle. Chaque processus a sa propre pile (WSN : pas approprié à cause des contraintes mémoire) et le context-switching induit un overhead significatif (WSN : consommation de CPU/énergie + risque de perte de données au niveau du capteur et de la radio).

Programmation basée sur les événements : Insère la nature réactive des noeuds WSN dans le modèle de programmation.



Principes de fonctionnement :
– boucle d'attente : ne fait rien, va au repos si nécessaire

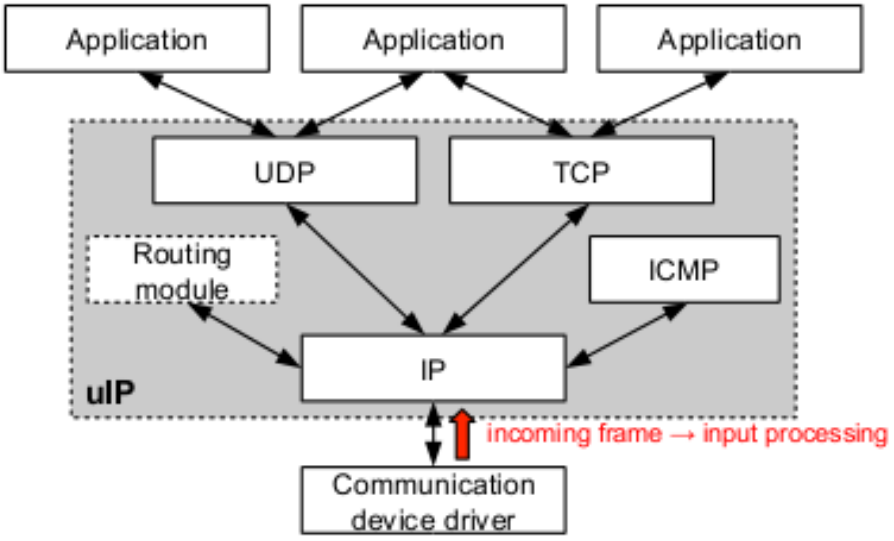
- gestionnaire d'événement : traite rapidement l'event entrant (donnée du capteur dispo, paquet entrant, ...), stocke les infos requises pour les traiter plus tard).
- un gestionnaire d'event peut interrompre tout code normal
- un gestionnaire d'event ne peut pas interrompre un autre gestionnaire (nécessiterait des coûts de context-switch - sauver piles et registres)
- le gestionnaire d'event tourne jusqu'à son achevement.

Il y a ainsi seulement deux contextes d'exécution :

- un pour les gestionnaires d'event à temps critique (sans interrupt)
- un pour le code normal (peut être interrompu)

6.7.3 Pile IP - uIP

Introdution : On a longtemps cru que IP était trop complexe et trop lourd pour être utilisé dans les petits systèmes embarqués en réseau (MCU limité...). Cette conviction a changé en 2001 avec la première version d'un prototype d'une pile IP complète pour un MCU 8-bits (inclus IP, ICMP, TCP, UDP).



Principes de fonctionnement : Arrivée de fenêtres de traitement du *Communication device driver*, et de paquets des applications. Réalise un traitement périodique.

- Traitement d'une entrée IP :
  - vérifie la version (supporte v4/v6)
  - vérifie la taille du header IP vis-à-vis de la taille de la frame reçue
  - vérifie le flag de fragmentation (pour v4) : si c'est un fragment alors il faut le copier dans le buffer de défragmentation, si ce buffer contient un datagramme IP complet alors continuer le traitement (pour v6 voir headers d'extension)
  - vérifie l'adresse source : jette les adresses illégales (source=broad/multicast)
  - vérifie l'adresse destination : distinction entre livraison locale et forwarding
  - vérifie la checksum du header (v4)
  - traite les headers d'extension (v6)
- Traitement d'une entrée ICMP :
  - pour v4 : uIP supporte seulement les message ICMP echo et va répondre avec un echo de réponse ICMP correspondant.
  - pour v6 : uIP supporte la découverte de voisins (messages NS/NA), la découverte de router (messages RS/RA), la détection d'adresse dupliquée, ...
- Traitement d'une entrée UDP :
  - vérifie la checksum UDP
  - effectue le demultiplexing : délivre à l'application destination selon le port UDP.
  - maintient une liste des applications et des ports associés.
- Traitement d'une entrée TCP : diffère fortement de la pile BSD UNIX
  - vérifie la checksum TCP
  - vérifie ports et IP src/dst vis-à-vis de la liste des connexions actives : compare le num de séq avec celui attendu, s'ils diffèrent on drop le segment (pas assez de mémoire pour stocker les segments mélangés), met à jour le RTT estimé, agit selon l'état de la machine à états TCP.

  - si aucune connexion active et le segment a un flag SYN, alors vérifie la liste des ports en écoute, se rappele du num de séq de l'envoyeur et l'option MSS (max size segm) de l'envoyeur (si présent) et envoit un segment TCP avec les flags SYN et ACK.
- TCP sans sliding window :
  - le mécanisme de sliding window permet d'envoyer les paquets multiples en pipeline à un certain temps et avec une meilleure efficacité. débit $\leq$ W/RTT
  - inconvénient : une *sliding window* nécessite beaucoup de mémoire.
  - uIP n'utilise pas les sliding window. Envoie un seul paquet à la fois, n'affecte pas l'interopérabilité ou les standards de conformité, affecte l'efficacité.
- Forwarding de paquet IP :
  - si l'adresse de destination d'un datagramme reçu n'est pas local
  - décrémente le TTL, vérifie ≥ 1
  - recalcule la checksum du header
  - pour chaque paquet, demande au module de routage de chercher l'adresse IP de destination : uIP ne spécifie pas de mécanisme de routage spécifique (peut être une table de destination, une table/arbre de préfixe, une table de hashage, un cache avec des résultats récents, ...), autorise uIP à être associé avec tout mécanisme de routage possible.

6.7.4 Cas d'étude : TinyOS

Voir cours :P