Chapitre 06 : Contrôle de trafic et QoS dans les réseaux IP

Partie II

« Mécanismes de contrôle de Trafic Opérant au Niveau Paquet »

Contrôle de trafic et QoS dans les réseaux IP

1. Applications
2. Mécanismes de contrôle de trafic de au niveau Paquet
   - ⇨ Service Best-effort
   - ⇨ Service de débit maximum garantie
   - ⇨ Service de débit minimum garantie
   - ⇨ Garanties de délai
3. Services Normalisés
4. Contrôle de trafic d'Intra-domaine
5. Contrôle de trafic d'Inter-domaine
6. Études de cas

Contrôle de trafic et QoS dans les réseaux IP

- Introduction :
  - Dans ce chapitre, on va voir les mécanismes qu'il faut mettre en place pour contrôler le trafic à l'intérieur du routeur
  - Qu'est ce qu'on peut mettre en place au niveau du routeur pour répondre au mieux aux besoins de ces applications
  - On va présenter ces mécanismes en **quatre parties** et on va voir à chaque fois les mécanismes qu'il faut mettre en place pour répondre à un besoin particulier.
    - Donc, on va partir d'un besoin et voir qu'est ce qu'on peut mettre en place pour répondre à ce besoin

Contrôle de trafic et QoS dans les réseaux IP

1. Applications
2. Mécanismes de contrôle de trafic de au niveau paquet
   1. Service Best-effort
   2. Service de débit maximum garantie
   3. Service de débit minimum garantie
   4. Garanties de délai
3. Services Normalisés
4. Contrôle de trafic d'Intra-domaine
5. Contrôle de trafic d'Inter-domaine
6. Études de cas

Contrôle de trafic et QoS dans les réseaux IP

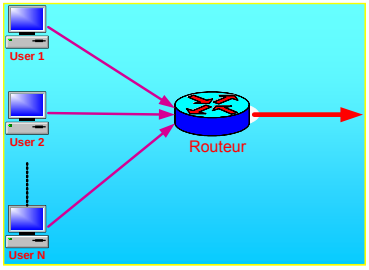
1. Applications
2. Mécanismes de contrôle de trafic au niveau paquet
    1. Service Best-effort
    2. Service de débit maximum garantie
    3. Service de débit minimum garantie
    4. Garanties de délai
3. Services Normalisés
4. Contrôle de trafic d'Intra-domaine
5. Contrôle de trafic d'Inter-domaine
6. Études de cas

Caractérisation du service Best-Effort

- Quel est le but du service Best-Effort ?
    - C'est un service ou on demande au réseau de faire son possible pour acheminer l'information à sa destination.
    - Faire son possible, ce n'est pas une définition suffisante pour tester un fournisseur d'accès Internet pour savoir s'il nous fournit un **`bon service Best-Effort.`**
- Comment caractériser un service **`Best-Effort`** :
    - C'est lorsqu'on essaye d'avoir un service pour lequel, **le réseau fournit un service équitable à tous ses utilisateurs** de façon à ce que si il y a **N utilisateurs** qui utilisent le réseau à moment donné, il reçoivent chacun une fraction **1/N du débit disponible**
    - Donc, l'idée est de répartir l'ensemble des **capacités du réseau** de façon équitable entre tous les utilisateurs du réseau.
    - Comment faire ça ?

Caractérisation du service Best-Effort

- Définition d'équité pour un lien simple
  - Sur une seule ligne, c'est relativement facile de parler d'équité entre utilisateurs



N utilisateurs **1/N du débit disponible**

Caractériser le service Best-effort

- *Définition d'équité pour un réseau complet :*
  - Si maintenant, on essaye étendre cette définition à un réseau avec un plus grand nombre d'utilisateurs qui communiquent ***dans tous les sens***, on peut plus simplement dire qu'on va prendre le débit du réseau et le diviser par N.
    - En effet, un utilisateur ADSL et un autre Dial-Up n'obtiendront pas le même débit (donc le service).
  - Questions :
    - Comment on peut faire pour leur fournir un service équitable ?
    - Comment caractériser l'équité dans un réseau hétérogène, dans lequel, il y a aussi des utilisateurs hétérogènes.

Caractériser le service Best-effort

- Définitions possibles de l'équité dans un réseau Hétérogène :

1. On peut redéfinir l'équité en cherchant l'équité qui permet de **`maximiser le débit`** fourni par le réseau

   ⇔ Faire en sorte que chaque utilisateur reçoit le plus de débit possible

2. Ou bien définir l'équité qui permet de ***`maximiser l'utilisation des ressources`*** du réseau

   - En effet, en tant que fournisseur de réseau, ce qui nous intéresse ce n'est pas de fournir le plus de débit, mais de mieux gérer le réseau (⇔ mieux charger les lignes du réseau)

     ⇔ ***`maximiser l'utilisation des ressources du réseau`***

Equité Max-Min

- Définition de l'Équité MAX-MIN pour un réseau
  - Le principe est d'avoir un mécanisme d'allocation de débit sur les différentes sources du réseau.
  - Cette allocation sera une allocation Max-Min si il maximise le débit aux ressources qui reçoivent le moins.
    - ⇔ **L'objectif est que la source qui consomme et qui reçoit le moins de débit qu'il en reçoive le plus possible.**
    - ⇔ **Donc, l'objectif de l'équité Max-Min est de fournir le plus de débit à l'utilisateur le plus pauvre du réseau de façon à satisfaire le plus possible l'utilisateur le plus pauvre, et puis en corollaire, on va essayer de satisfaire le plus possible tous les utilisateurs**

Equité max-min : algorithme

- Comment déterminer une allocation Max-Min ?
  1. Au départ, on alloue 0 Mbps à chaque source
  2. On incrémente ensuite
     - *Le débit « a bande passante » alloué à chaque source jusqu'à saturation d'un lien (arrivée d'une congestion sur la liaison)*
     - *Les débits des sources utilisant les liens saturés sont fixées*
  3. Après chaque saturation d'un lien
     - *On continue seulement à incrémenter le débit des sources encore non fixées*

Équité Max-Min

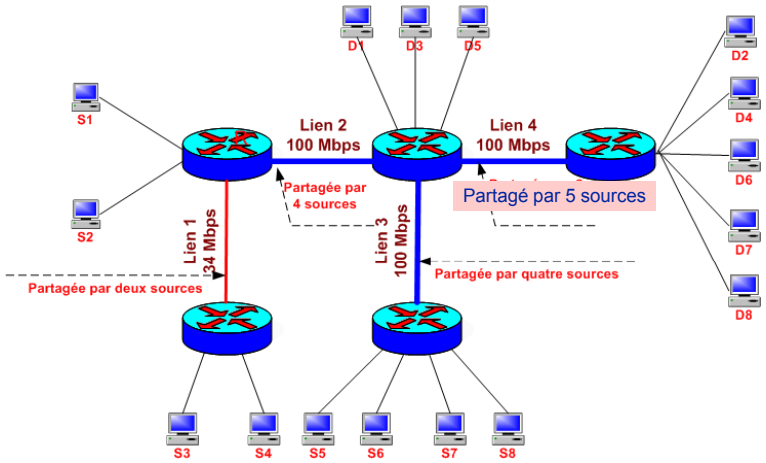
Exercice

Exercice : algorithme équité max-min

Objectif : Allocation Max-Min : ***maximiser le débit qui est reçu par les sources les plus pauvres***

S1 -> D1 : via Lien 2
S2 -> D2 : via Lien 2 et Lien 4
S3 -> D3 : via Lien 1 et Lien 2
S4 -> D4 : via Lien 1, Lien 2 et Lien 4

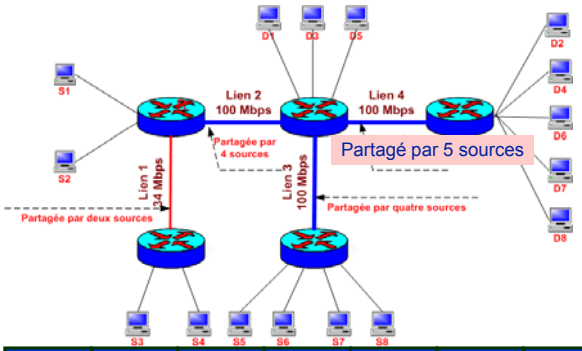
S5 -> D5 : via Lien 3
S6 -> D6 : via Lien 3 et Lien 4
S7 -> D7 : via Lien 3 et Lien 4
S8 -> D8 : via Lien 3 et Lien 4



Exercice équité Max-Min - solution

Exercice – Tra. 1 -

Au départ : Toutes les sources sont à égalité (⇔ débit de 0 Mbits/s par source)

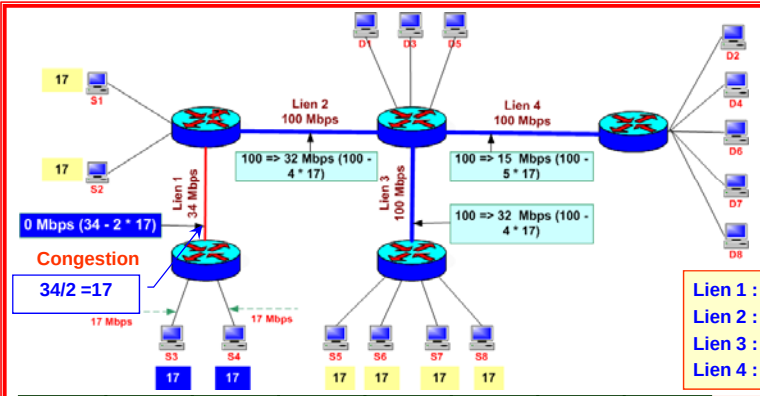


S1 -> D1 : via Lien 2
S2 -> D2 : via Lien 2 et Lien 4
S3 -> D3 : via Lien 1 et Lien 2
S4 -> D4 : via Lien 1, Lien 2 et Lien 4
S5 -> D5 : via Lien 3
S6 -> D6 : via Lien 3 et Lien 4
S7 -> D7 : via Lien 3 et Lien 4
S8 -> D8 : via Lien 3 et Lien 4

Lien 1 : S3, S4
Lien 2 : S1, S2, S3, S4
Lien 3 : S5, S6, S7, S8
Lien 4 : S2, S4, S6, S7, S8

S1	S2	S3	S4	S5	S6	S7	S8

Exercice – Tra. 2 -

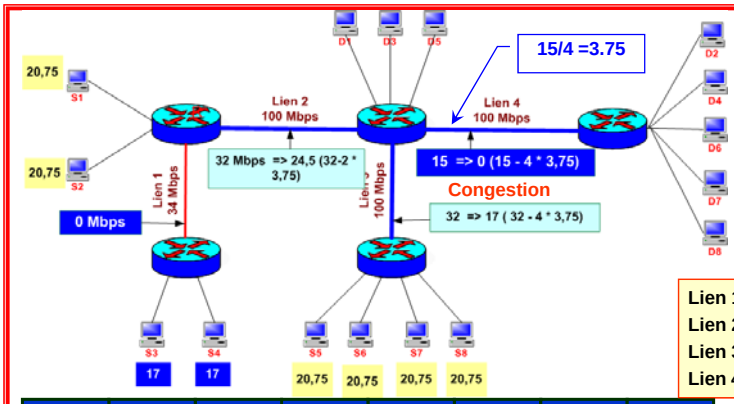


Lien 1 : S1, S2
Lien 2 : S1, S2, S3 (fixé), S4
Lien 3 : S5, S6, S7, S8
Lien 4 : S2, S4(fixé), S6, S7, S8

S1	S2	S3	S4	S5	S6	S7	S8
17	17	17	17	17	17	17	17

S1 -> D1 : via Lien 2
S2 -> D2 : via Lien 2 et Lien 4
S3 -> D3 : via Lien 1 et Lien 2
S4 -> D4 : via Lien 1, Lien 2 et Lien 4
S5 -> D5 : via Lien 3
S6 -> D6 : via Lien 3 et Lien 4
S7 -> D7 : via Lien 3 et Lien 4
S8 -> D8 : via Lien 3 et Lien 4

Exercice – Tra. 3 -

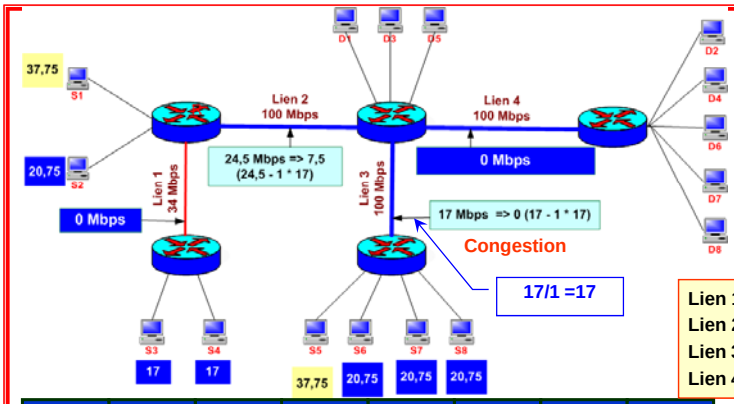


Lien 1 : S1, S2
Lien 2 : S1, S2, S3, S4
Lien 3 : S5, S6, S7, S8
Lien 4 : S2 , S4, S6, S7, S8

S1	S2	S3	S4	S5	S6	S7	S8
17	17	17	17	17	17	17	17
20.75	20.75			20.75	20.75	20.75	20.75

S1 -> D1 : via Lien 2
S2 -> D2 : via Lien 2 et Lien 4
S3 -> D3 : via Lien 1 et Lien 2
S4 -> D4 : via Lien 1, Lien 2 et Lien 4
S5 -> D5 : via Lien 3
S6 -> D6 : via Lien 3 et Lien 4
S7 -> D7 : via Lien 3 et Lien 4
S8 -> D8 : via Lien 3 et Lien 4

Exercice – Tra. 4 -

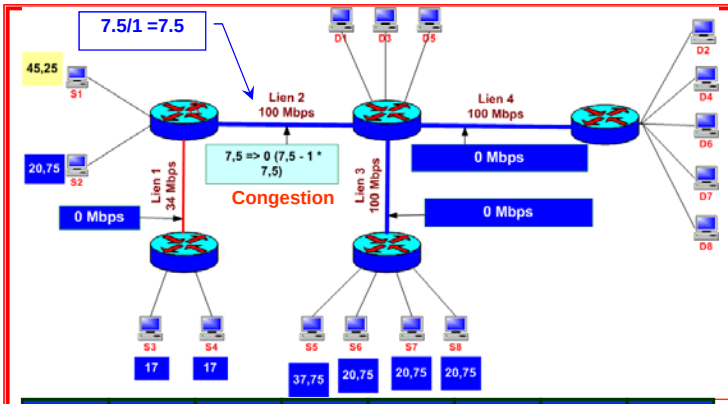


Lien 1 : S1, S2
Lien 2 : S1, S2, S3, S4
Lien 3 : S5, S6, S7, S8
Lien 4 : S2 , S4, S6, S7, S8

S1	S2	S3	S4	S5	S6	S7	S8
17	17	17	17	17	17	17	17
20.75	20.75			20.75	20.75	20.75	20.75
37.75				37.75			

S1 -> D1 : via Lien 2
S2 -> D2 : via Lien 2 et Lien 4
S3 -> D3 : via Lien 1 et Lien 2
S4 -> D4 : via Lien 1, Lien 2 et Lien 4
S5 -> D5 : via Lien 3
S6 -> D6 : via Lien 3 et Lien 4
S7 -> D7 : via Lien 3 et Lien 4
S8 -> D8 : via Lien 3 et Lien 4

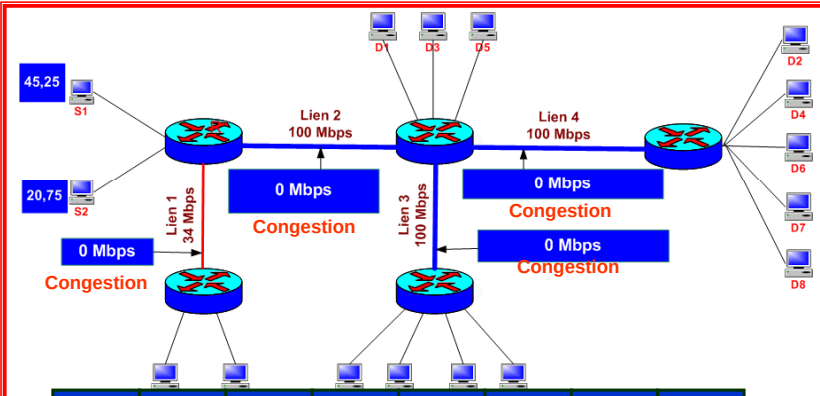
Exercice – Tra. 5 -



S1	S2	S3	S4	S5	S6	S7	S8
17	17	17	17	17	17	17	17
20.75	20.75			20.75	20.75	20.75	20.75
37.75				37.75			
45.25							

S1 -> D1 : via Lien 2
S2 -> D2 : via Lien 2 et Lien 4
S3 -> D3 : via Lien 1 et Lien 2
S4 -> D4 : via Lien 1, Lien 2 et Lien 4
S5 -> D5 : via Lien 3
S6 -> D6 : via Lien 3 et Lien 4
S7 -> D7 : via Lien 3 et Lien 4
S8 -> D8 : via Lien 3 et Lien 4

Exercice – Tra. 6- Bilan -



S1 : 42,25 Mbps
S2 : 20,75 Mbps
S3 : 17 Mbps
S4 : 17 Mbps
S5 : 37,75 Mbps
S6 : 20,75 Mbps
S7 : 20,75 Mbps
S8 : 20,75 Mbps

S1	S2	S3	S4	S5	S6	S7	S8
17	17	17	17	17	17	17	17
20.75	20.75			20.75	20.75	20.75	20.75
37.75				37.75			
45.25							

Comment fournir l'équité Max-Min dans un réseau TCP?

- **L'équité MAX-MIN dans un réseaux TCP/IP :**
  - Est un objectif idéal, difficile à atteindre dans la pratique (Du fait des problèmes de congestion).
  - Objectif : est de s'y approcher le plus possible

Comment fournir l'équité Max-Min dans un réseau TCP?

- **Dans un réseau TCP/IP, l'équité dépendra de 2 facteurs**

1. Traitement de paquets à l'intérieur de routeur :
   - L'élément important sera la façon dont on **va jeter les paquets dans le routeur**.
   - **Pourquoi ?** En effet, l'élément pour indiquer la congestion dans **TCP c'est la perte des paquets**.
     - ⇔ Lorsqu'on **va perdre des paquets**, on va influencer le comportement de TCP en cas de congestion. Donc, en **acceptant ou pas** des paquets, on va avoir une influence sur les sources TCP

⇔Quels mécanismes d'acceptation des paquets ?

⇔Quels sont les paquets détruits en cas de congestion ?

Comment fournir l'équité Max-Min dans un réseau TCP?

2. Mécanisme de contrôle de congestion implémentés dans les équipements d'extrémité :

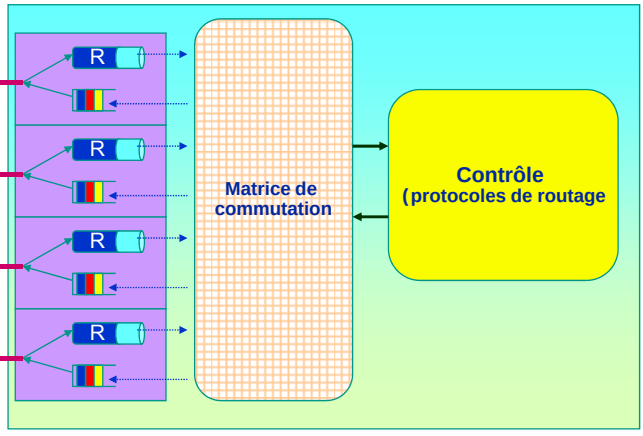
- L'équité va dépendre de contrôle de congestion utilisé par les sources.
- Dans TCP, le contrôle de congestion est implémenté dans toutes les sources.
  - Donc, si le réseau utilise exclusivement TCP, les sources vont s'adapter au réseau en direction d'une allocation Max-Min de débit.
- Par contre, si on a un réseau uniquement avec des applications UDP, ces applications ne vont pas s'adapter seules à la congestion du réseau (pas de contrôle de congestion).
  - Le seul contrôle de congestion qu'il y a au niveau **d'UDP c'est l'utilisateur.**

Routeur Simple – modèle de base

- Pour développer ces mécanismes de contrôle de trafic au niveau d'un routeur, on va considérer un modèle simple.
- L'organisation classique d'un tel routeur (Routeur avec 4 ports d'entrée sortie)

♦**Sur chaque port d'E/S, on va avoir deux mécanismes importants :**

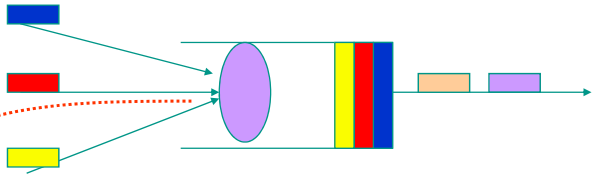
- Le routage sur les paquets qui rentrent pour déterminer quel sera le port de sortie
- Tous les mécanismes qui vont nous permettre de contrôler le trafic qui va être transmis sur la ligne de sortie



Routeur simple v1

- *Sur le **port de sortie d'un routeur**, on aura :*
  - La liaison sur laquelle, on transmet les paquets
  - En amont, on a **un buffer** qui va servir pour **stocker les paquets** qui sont en attente pour être transmis sur la ligne de sortie
  - Et en amont de ce buffer, on va avoir le **premier mécanisme de contrôle de trafic** :

mécanisme d'acceptation des paquets



- ***Algorithme d'Acceptation des paquets :***
  - Quand un paquet arrive d'un lien d'entrée, l'algorithme d'acceptation des paquets décide si ce paquet peut être accepté à l'intérieur de la file d'attente du routeur
  - Via ce mécanisme, on va décider quels sont les paquets qui vont être **accepté** ou **jetés**

Algorithmes Acceptation des paquets -1-

Mécanisme d'acceptation des paquets : Deux Questions se Posent?

A. *Première question : Quand supprimer un paquet ?*

⌦ ***Deux approches possibles :***

1. Quand le buffer des sortie du routeur est plein (⇔ lorsqu'on est obligé : approche simple qui se retrouve sur tous les routeurs
   ⌦ **Exemple : « Tail Drop »**
2. Deuxième Alternative : S'appuyer sur l'idée que dans un environnement TCP/IP, TCP cherche à s'adapter à la congestion.
   ⌦ **TCP** va chercher à charger les buffers du routeurs. Si les buffers commencent à augmenter dangereusement, c'est qu'il y des sources qui commencent à transmettre trop vite. **D**ans ce cas, il va falloir informer ces source pour qu'elles transmettent moins vite**.**
   ⌦ Pour informer les sources dans TCP, la seule solution qu'on a est de jeter des paquets. (⇔ Quand le **taux d'occupation** de la file devient trop important)
   ⌦ Exemple **:** **RED** = **R**andom **E**arly **D**etection

Algorithmes Acceptation des paquets -2-

A. *Deuxième question : Quel paquet supprimer ?* ***Plusieurs réponses possibles***

1. ***Première approche*** **: est de jeter le paquet qui arrive (car on l'a pas encore mis dans le buffer, c'est la plus simple de point de vue implémentation)**
   - Par contre, ce n'est pas la solution qui va donner la meilleur équité **Max-Min**, car si un paquet arrive et qu'on est en situation de congestion, ce n'est pas forcément le flux associé à ce paquet qui est **responsable de la congestion**

2. **Ou bien jeter un paquet *d'une autre connexion TCP* que celle pour laquelle, le paquet arrive** (par exemple de la connexion TCP qui est la plus responsable de la congestion du réseau).
   - Pour cela, il faut être **capable d'identifier les paquets** de la connexion qui est la plus responsable de la congestion

3. **Un autre mécanisme qui consiste à jeter le paquet *qui allait être transmis sur la ligne de sortie***

Algorithmes Acceptation des paquets -3-

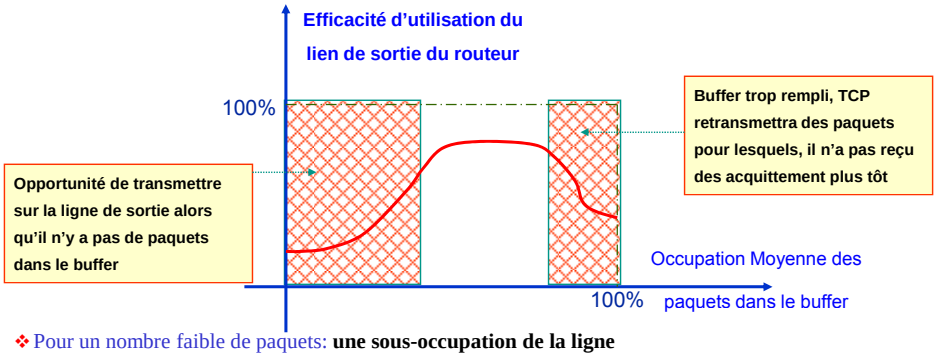
- Objectifs : Contrôler la quantité de paquets dans la file d'attente dans le but :

1. Supporter efficacement le trafic best effort
   - ⇔ Tendre vers une allocation Max-Min de débit (⇔ tendre vers une allocation Max-Min, donc équitable des buffers du routeur)

2. On voudrait bien au niveau du routeur avoir une certaine forme de protection entre les connexions TCP
   - Un flux ne doit pas empêcher les autres flux d'avoir des paquets dans les buffers du routeur
   - une connexion ftp ne doit pas empêcher le reste des applications d'utiliser un réseau WAN ou un étudiant ne puisse pas utiliser l'entièreté de la connexion à un réseau.

Algorithmes Acceptation des paquets -4-

3. Effectuer une bonne utilisation de la ligne de sortie du routeur ⇔ avoir une occupation des buffers du routeur qui n'est ni ***nulle*** ni ***complètement en saturation***



- Pour un nombre faible de paquets: **une sous-occupation de la ligne**
- Par contre, pour un nombre très important de paquets , et comme on est avec une majorité d'applications qui utilisent TCP qui cherchent à envoyer au maximum avant de recevoir des acquits. Dans le cas d'une congestion, on risque avoir le même paquet **qui est transmis plusieurs fois sur la ligne de sortie**
  (⇔ une baisse d'efficacité de l'utilisation de la ligne de sortie)

Algorithmes Acceptation des paquets

- Solutions : permettant de répondre partiellement à ces objetcifs :
  1. Mécanisme Tail-Drop
  2. Mécanisme RED : Radnom Early Detection

1. Mécanisme : Tail Drop

- L'implémentation la plus simple de l' Algorithmes Acceptation des paquets
- **Principe :**
  - Lorsqu'un paquet arrive dans **un buffer plein**, le paquet est tout simplement jeté (paquet discarding) (⇔ on regarde ***l'Occupation instantanée*** du buffer)
- **Avantages :**
  - **Facile à mettre en application** (implémenté dans tous les routeurs)
  - Peut limiter le nombre de pertes de paquet **dans le cas ou on a un buffer de taille importante**
- **Inconvénients :**
  - Aucune **distinction entre les différentes** connexions ou les différents flux
  - N'est pas la meilleure solution **« de point de vue performances »** pour le trafic TCP (car, quand le buffer est plein, il va rester ainsi quelques temps durant lequel, on va jeter des paquets TCP ⇔ on risque de jeter des **groupes de paquets d'une même connexion** TCP => mauvaises perf au niveau TCP, oscillation au niveau de TCP et qui sont dues à des oscillations au niveau des buffers)

2. Random Early Detection : RED « Détection préventive aléatoire » -1- -RFC 2309-

- **Objectifs fixés (date de 1993) :**
  1. Doit être simple à implanter dans un routeur
     - Avec une seule file d'attente par sortie
  2. Doit assurer un taux d' occupation faible (mais non nul) des files d' attente en sortie à fin offrir des délais relativement faibles aux applications interactives ,
     - **Exemple :** Si on fait passer des **paquets Telnet** dans un buffer qui est tout le temps rempli, ils vont devoir attendre de traverser tout le buffer avant leur transmission (⇔ vont avoir des délais importants). Par contre, si le buffer est peu rempli, le paquet Telnet passera plus rapidement
     - Un taux d'occupation faible « non nul » fournit une bonne utilisation des liens de sortie
  3. Doit supprimer les paquets de manière équitable entre toutes les connexions TCP qui se partagent la liaison de sortie :
     - En évitant de favoriser certains connexions plus que d'autres
     - Et sans avoir besoin à identifier explicitement les différentes connexions TCP

2. Random Early Detection : RED « Détection préventive aléatoire » -2-

4. Il faut permettre à TCP (⇔ le contrôle de congestion TCP) de **s'adapter facilement** aux pertes des paquets **provoqués par ce mécanisme**.

   ⇔ éviter de jeter de groupe de paquets d'une même connexion TCP, puisque TCP réagit de façon excessive lorsqu'on perd un groupe de paquets d'une même connexion

   ⇔ essayer avec RED de jeter des ***paquets individuellement*** plus tôt que de jeter des ***groupes de paquets***

2. Random Early Detection : RED « Détection préventive aléatoire » -3-

- Implémentation : Principes de base (⇔ répondre à deux question)

1. **Comment détecter une congestion ?**
   1. En mesurant le taux d'occupation moyenne du buffer du routeur (filtre passe-bas)
   2. On est en congestion lorsque le taux d'occupation moyenne dépasse un seuil fixe
      - En pratique, de l' ordre de 10 à 20% de sa capacité (paramètre de configuration)
      - Ceci nous permettrait de garder de la place dans les buffers en cas de congestion
2. **Que faire lorsqu'on détecte qu'on est en congestion ?**
   - **Suppression aléatoire des nouveaux paquets arrivés**

     ⊠ La probabilité de suppression devra croître en fonction **du niveau de congestion**

     (On sait que si on jette un paquet, ça forcera TCP à réduire son débit et on aura une probabilité de perte qui dépendra du niveau de congestion du routeur)

2. Random Early Detection : RED « Détection préventive aléatoire » -4-

■ **Pourquoi une suppression aléatoire ?**

1. On sait que de cette minière, la probabilité de jeter un groupe de paquets d'une même connexion TCP est très faible (une bonne chose pour TCP).

2. Ceci permet de jeter les paquets au prorata de débit utilisé par chacune des sources

⌧ **Justification : Exemple**

- Une connexion TCP qui correspond à **Ftp qui utilise 95%** de débit de la liaison de sortie
- Une connexion **Telnet qui utilise 5%** de débit de la liaison de sortie
- Les buffers de routeurs vont arriver en congestion, dans ce cas avec 95% de volume venant de ftp, on va avoir plus souvent arriver des paquets ftp que des paquets telnet.
- Chaque fois un paquet arrive, on va décider de **façon probabiliste** si on l'accepte ou si on le jette. Le paquet **telnet** arrive **peu souvent,** il y a peu de chance qu'il soit jeté. Par contre des **paquets ftp**, il y en a tout le temps des paquets qui arrivent, donc très rapidement on va finir par en jeter un.
- Donc on va jeter les paquets au prorata des ressources qui sont consommées par chacune des applications sans avoir besoin d'identifier leurs connexions et sans avoir besoin de mesurer leurs débits

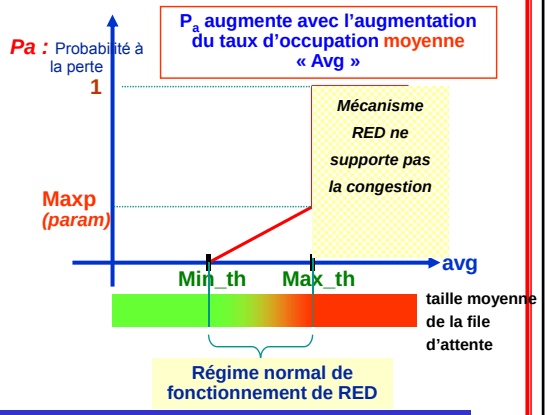
2. Random Early Detection : RED « Détection préventive aléatoire » -5-

■ Implémentation du RED

Filtre :
Avg = (Avg * (1-wq)) + (Occupation du buffer * wq)
Avec : wq < 1 (en général, il est de la forme ½)

// A chaque arrivée d'un paquet
1. Calculer le taux d'occupation moyen du buffer « Avg »
2. Acceptation du paquet
Si (Avg < min_th)
// Pas de congestion
Accepter le paquet
Sinon, Si (Min_th ≤ Avg ≤ Max_th)
• Calcule une probabilité Pa(avg)
• **Détruire le paquet avec la proba. Pa**
Sinon Si (avg ≥ Max_th)
Détruire le paquet

$P_a = Maxp(Avg - Min\_th) / (Max\_th - Min\_th)$



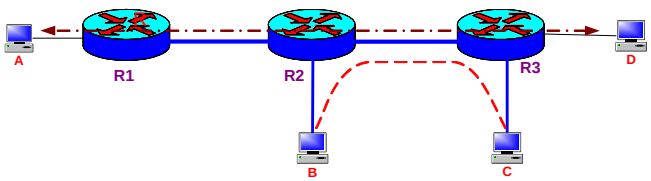
http://www.icir.org/floyd/papers/early.twocolumn.pdf
http://www.linux-france.org/prj/inetdoc/guides/Advanced-routing-Howto/lartc.adv-qdisc.red.html

Autres solutions pour implémenter le contrôle de congestion

TCP « ⇔ Sans Destruction « suppression » des paquets »

Destruction « suppression » des paquets

- **Problème posé : Dans TCP, la perte des paquets ⇔ seul indicateur de congestion.**
  - **Le problème est de jeter un paquet qui peut avoir déjà consommé des ressources**
- **Exemple :**
  - Les paquets de la connexion TCP de **A** vers **D** vont être en concurrence avec ceux de la connexion **TCP** de B vers C (liaison R2–R3).
  - Lorsqu'on a des paquets de **A** vers **D** qui arrivent au niveau de R2 et qui risquent d'être jetés en R2 pare ce que la liaison R2–R3 est trop chargée, ces paquets ont déjà consommées des ressources sur la liaison R1–R2.



- Ce n'est pas très optimal, par ce que cette connexion aurait pu être utilisée par d'autres paquets

❖ Donc, la question **est de savoir si on peut fournir une équité MAX-MIN** sans jeter des paquets (⇔ contrôler la congestion sans nécessairement jeter des paquets ?)

D'autres alternatives au contrôle de congestion TCP

- Solution DECBit (Destination Experiencing Congestion Bit, implémenté dans les protocoles ISO et DECNET Classe 5 : réf : http://users.skynet.be/baptiste.buxant/tfe.pdf)
- Idée de base de DCBit : (ne nécessite pas de supprimer des paquets) :
  - Les routeurs marquent un paquet congestionné par une étiquette
  - la destination renvoie des indications de congestion dans les acquittements (**acks**)
  - Implémenté par DEC dans des protocoles ISO

♦ **Implémentations :**

1. **Solution Frame Relay (Relais de trame)**
   - **FECN :** congestion indiquée dans la direction descendante dans le VC
   - **BECN :** congestion indiquée dans la direction ascendante dans le VC
2. **Solution ATM**
   - **ABR :** Available Bit Rate
     - Bit-Based Congestion Indication (EFCI bit)
     - Explicit Rate Congestion Indication (cellules de RM)

Notification Explicite de congestion (ECN) en TCP-1- « RFCs 2481, 2884, 3168, 3540 »

- Est-ce qu'on peut adopter ces mécanismes dans TCP.
- Deux Problèmes à résoudre :

1. Il faut qu'on puisse détecter la congestion avant qu'elle se produise

   ⇔Marquer les paquets pour indiquer qu'il y a de la congestion.

   - ***Difficile à mettre en place avec « Tail-Drop »*** destruction obligatoire (file pleine) à partir du moment ou le buffer est plein
   - ***Plus facile avec RED***, à partir du moment ou le buffer n'est jamais plein (on pourra toujours conserver des paquets qu'on ***aura marqué*** pour le congestion)
2. Comment informer explicitement les sources « notification de congestion » de la congestion du réseaux

Notification Explicite de congestion en TCP-2-

◆ Deux Solutions proposées pour informer la source de la congestion :

1. **Informer directement la source**
   - En transmettant un message ICMP particulier vers la source (paquet de taille 40 octets +) doit être généré. Le paquet en entier doit être transmis dans un réseau potentiellement congestionné
   - Un message ICMP de ce type existe, mais n'a jamais été utilisé dans la pratique **(source Quench)**
2. **Informer le destinataire, puis la source en retour**
   - **A la place, on va essayer d'informer la destination en marquant le paquet IP qui a traversé le routeur en congestion.**
     - On va devoir modifier un **bit** à l'intérieur **d'un paquet IP** pour servir à marquer de congestion.
   - **Comment ?**

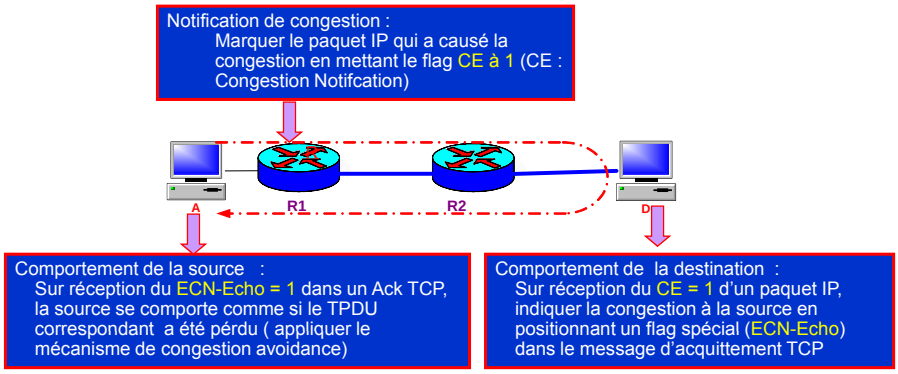
Notification Explicite de congestion en TCP-3-

- **On va modifier le comportement de TCP aux sources et aux destinations et on va modifier la façon dont les routeurs vont fonctionner**

1. **A la source :** Les sources TCP vont envoyer leurs paquets
2. **Au niveau d'un routeur :** lorsqu'on se rend compte qu'un routeur est en congestion, on va permettre au routeur **de marquer le paquet** plutôt **que de le jeter** lorsque la congestion est détectée.
   - Pour ça, on va ajouter un bit dans l'en-tête d'un paquet IP (bit **CE : Congestion Experienced**)
3. **Au niveau de la destination :** Le paquet va arriver à destination, qui se rend compte que le bit **CE est mis à vrai** => il va demander à la source de réduire son débit.
   - Donc lorsque la destination reçoit un paquet avec le bit **CE = 1,** il va utiliser un **flag particulier** dans les acquittements TCP qu'il va envoyer (bit **ECN_ECHO**) pour signaler à la source de l'arrivée d'une congestion sur le paquet envoyé.
4. **Lorsque la source reçoit l'acquit avec ECN_ECHO positionné, elle va réagir comme si le paquet avait été perdu.**
   - ⇔en appliquant la congestion avoidance (⇔ **en réduisant sa fenêtre de congestion par deux**, et en recommençant à transmettre)

Notification Explicite de Congestion (ECN) dans TCP -4-

- Intégration dans TCP : principe de solution



ECN – Perte des acks –1-

- Problèmes Potentiels :

1. 99,99% (?) des Sources TCP « Destination » ne supportent pas l'extension ECN

2. Que se produit si l'acquittement de ECN-ECHO est perdu ?
   - En effet : l'acquittement envoyé par la destination et qui contient le bit ECN-ECHO à 1 n'est pas envoyé de façon fiable à la source.
   - Donc, si il est perdu, la source ne aurait pas été au courant de la congestion.
   - Remarque :
     - Ceci n'empêchera pas TCP de fonctionner, puisque dans TCP les **acquits sont commutatifs**, donc le prochain acquit envoyé à la source par la destination permettra à la source de récupérer son fonctionnement normal, **mais n'aurait pas été au courant de la congestion** ⇔ n'aurait pas réduit son débit d'émission, et on aurait fini par jeter des paquets inutilement.

ECN – Perte des acks-2-

- Solution : il faut rendre la transmission de ECN-ECHO fiable de la destination vers la source.

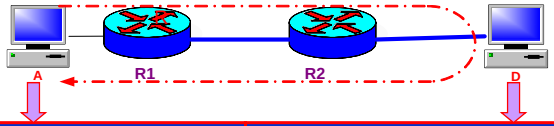
1.Première Approche

- Pour ça, a on pourra dire à la destination d'envoyer N fois le flag ECN-ECHO à la source, mais dire à la source de ne réagir qu'une seule fois.
- Le problème est la bonne valeur de N

ECN – Perte des acks-3-

1.Deuxième approche : A la place on va permettre à la source d'acquitter qu'elle bien vu le bit ECN-ECHO . L'idée est la suivante :

- La destination a reçu un paquet IP avec **CE = 1,** Il informe la source. Pour cela, on va mettre le bit **ECN-ECHO à vrai** dans tous les acquits qu'on va émettre vers la source
- On va laisser ce bit **ECN-ECHO à vrai** jusqu'à ce que la source confirme qu'elle bien reçu le bit **ECN-ECHO**
- On ajoute un autre flag dans l'en-tête des paquets envoyés par la source. Ce flag est le **CWR (Congestion Window Review)** qui sera utilisé pour acquitter **ECN-ECHO** (N.B : Comme ce bit se trouve à l'intérieur **des segments données envoyés par la source**, il sera transmis de **façon fiable vers la destination**)
- La destination, quand il verra un paquet revenir avec un flag **CWR mis à vrai**, elle saura que son **ECN-ECHO** a bien été transmis et reçu par la source



Comportement de l'émetteur : Sur réception du ECN-Echo = 1 dans un Ack TCP,
1. Exécuter le mécanisme de congestion avoidance
2. Et positionner le fag CWR dans le TCP PDU suicvant

Comportement du receveur :
Sur réception du CE = 1 d'un paquet IP, indiquer la congestion à la source en positionnant un flag spécial (ECN-Echo) dans tous les messages d'acquittement (TCP Ack) jusqu'au à ce qu'un TCP PDU avec (CWR=1) soit reçu par le receveur

Déploiement d' ECN

Dans les terminaux

Dans les routeurs

Déploiement d' ECN dans les terminaux

- Comment faire supporter ECN par les systèmes d'extrémité ?

1. Système utilisant TCP :
   - TCP doit être modifié pour supporter les nouveaux flags
   - Pendant la phase d'établissement de connexion, l'utilisation d'ECN doit être négociée pendant la phase d'établissement de connexion en trois étapes
     - → Si tous les systèmes d'extrémité supportent ECN, il peut être utilisé
     - → Si un des systèmes d'extrémité ne supporte pas ECN, c'est le mécanisme de contrôle de congestion « normal » qui est appliqué
2. Systèmes utilisant autre protocoles de transport :
   - Un travail supplémentaire nécessaire pour adapter les mécanismes de contrôle de congestion pour supporter l'ECN

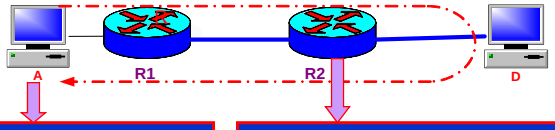
Déploiement d' ECN dans les routeurs

■ **Pour travailler correctement, un routeur devrait pouvoir distinguer entre :**

→ **Un paquet appartenant à des connexions TCP qui sont normales pour lesquelles, il faut jeter des paquets en cas de congestion**

→ **Et un paquet appartenant à des connexions TCP ou il y de support de ECN au niveau de source et de destination**

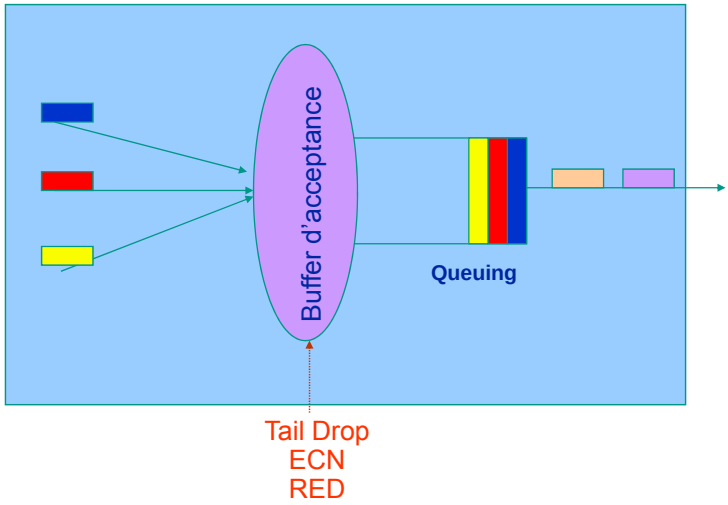
- **Solution** : ajouter un bit au niveau de l'en-tête IP (bit **ECT : Explicit Congestion Transport**) qui identifie si **la source supporte explicitement** le mécanisme **ECN** ou Non
- Au niveau de routeur, lorsqu'on va détecter la congestion, on va regarder la valeur du bit ECT
  - Si **ECT = « vrai »,** on va simplement **marquer le paquet** conforment au **ECN** (marquage en cas de congestion)
  - Si **ECT =« Faux »,** on va **jeter le paquet,** par ce que c'est la seule façon d'informer la source



Emetteur supportant ECN (ECN-Capable)
Si le receveur supporte aussi ECN
ECT = 1 dans tous les paquets IP à destination du receveur

Dans le cas d'une congestion
Si (ECT = 1)
Marquer le paquet IP qui a causé la congestion en mettant le flage CE = 1 (CE : Congestion Experienced)
Si (ECT n'est pas positionné)
Jeter le paquet qui a causé la congestion

Architecture du routeur pour le trafic Best-Effort « Bilan »



Inconvénients des solutions basées sur TCP

- Inconvénients de Tail Drop, RED, ECN
  1. **Equité Max-Min n'est pas toujours réalisée**
     - Les connexions **TCP** avec des **RTT** importants sont plus pénalisés comparés à aux connexions **TCP** qui ont des **RTT** plus faibles
  2. **ECN, RED et Tail-Drop Supposent que toutes les sources sont coopératives ⇔ réagissent toute à la congestion**
     - Les applications **UDP** n'intègrent pas le contrôle de congestion
     - Certains implémentations **TCP** peuvent ne pas mettre en application correctement les **mécanismes de contrôle de congestion**
- Pouvons nous fournir une équité **« Max-Min »** indépendamment de contrôle de congestion TCP's?

Les autres solutions pour l'équité Max-Min « le Best-Effort »

indépendamment de contrôle de congestion TCP

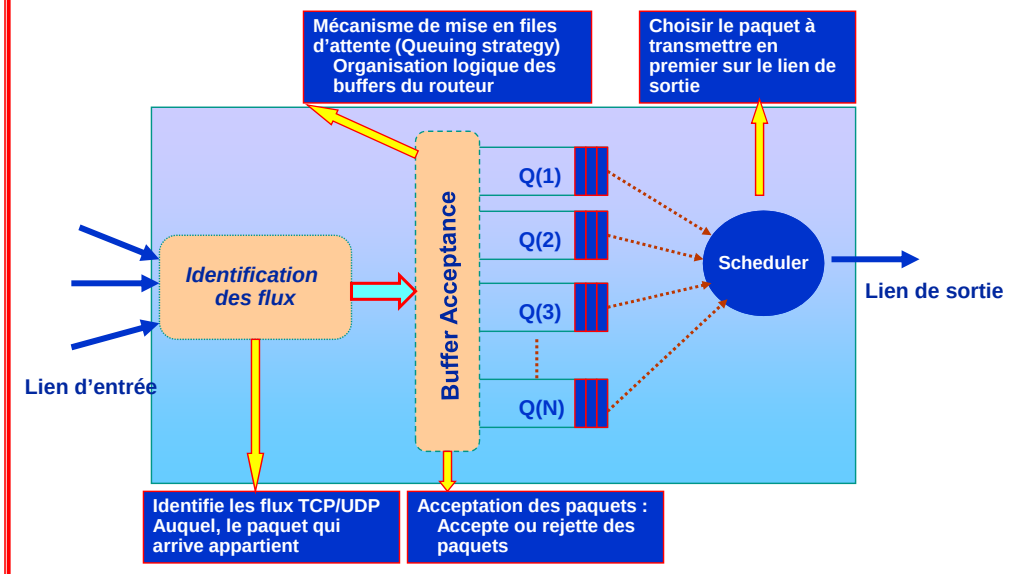
Principes d'une solution

- **SUR CHAQUE LIAISON DE SORTIE :**
  1. Diviser les buffers en N queues logiques
  2. Identifier la connexion TCP ou le flux UDP auquel chaque paquet appartient
     - Les paquets appartenant à un flux donné sont placés dans une même file d'attente
  3. S' il y a F flux (ou connexions) à un instant t ; s'assurer que chaque flux utilise 1/F de la largeur de bande disponible

Principes d'une solution

- **Pour réaliser ceci, deux modifications sur notre routeur** :
  1. ***Pouvoir identifier une connexion TCP ou bien un flux UDP auquel***, un paquet appartient
     - identification du flux en amont du buffer de sortie
     - Après identification, passer le flux dans la queue logique associée à cette connexion.
  2. ***Il faut un mécanisme supplémentaire « Scheduler ⇔ Ordonnanceur »*** qui permet de décider du paquet qu'on va servir à un moment donné
- **En travaillant sur la répartitions des flux dans les buffers et sur l'ordonnanceur, on doit pouvoir obtenir les objectifs qu'on s'est fixés précédemment.**

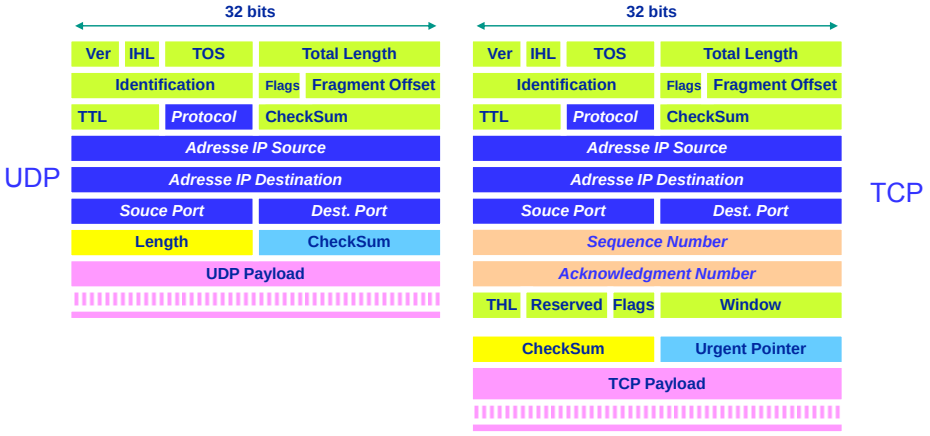
Routeur Best-Effort – Version améliorée



Comment identifier les flux TCP / UDP?

- Les flux du niveau quatre sont identifiés par le quintuple (5-uple)
  1. Adresse IP de la source,
  2. Adresse IP de destination,
  3. Type de protocole,
  4. N° de port source,
  5. N° de port de destination

  Identification d'une connexion
- Ces informations sont localisées dans les mêmes positions dans les en-têtes TCP et UDP



Ordonnancement des flux Best-Effort

- **Objectifs d'un tel Scheduler : il doit être capable de :**

1. **Fournir une distribution équitable de débit entre les flux actifs pour supporter l'équité Max-Min à l'échelle du réseau**
   - L'équité ne devrait pas dépendre du comportement des mécanismes de contrôle de congestion dans les systèmes d'extrémité (⇔ **sans faire des hypothèses sur les sources**)
   - **L'Ordonnanceur** doit garantir une **distribution correcte** « équitable » de la capacité de la liaison de de sortie entre les flux
   - ⇔ Les **mécanismes d'acceptation des paquets** dans les buffers **soient équitables également**
2. **Assurer l'isolement entre les flux**
   - Un mauvais comportement possible d'un flux ne devrait pas se traduire par une consommation d'une grande partie de la bande passante disponible
3. **Implantable aux débits élevés**

IMPLEMENTATIONS DE L'ORDONNANCEUR

Processor Sharing (PS) « Partage du processeur »

- **PS est un concept mathématique intéressant par ce que sur la base de ce concept, tous les autres mécanismes vont être construits**

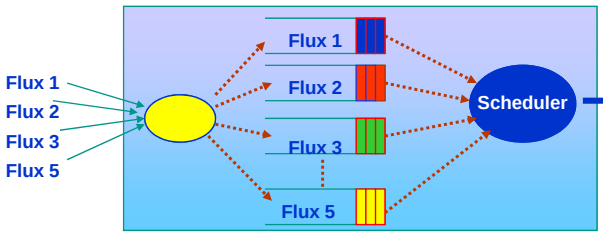
- **IDEE DE BASE :**

1. Construire un Scheduler **« idéal »** dans lequel, on va servir chaque queue **comme si la queue contenait un fluide** (⇔ on suppose que dans chaque queue, il y a une liquide)

⇨ A chaque instant, on regarde une queue s'elle contient de liquide et on ouvrira les vannes de chacune des queues de **façon équitable** au niveau de la sortie

⇨ A tout instant, on servira la **Queue j** à un débit donné par la relation :

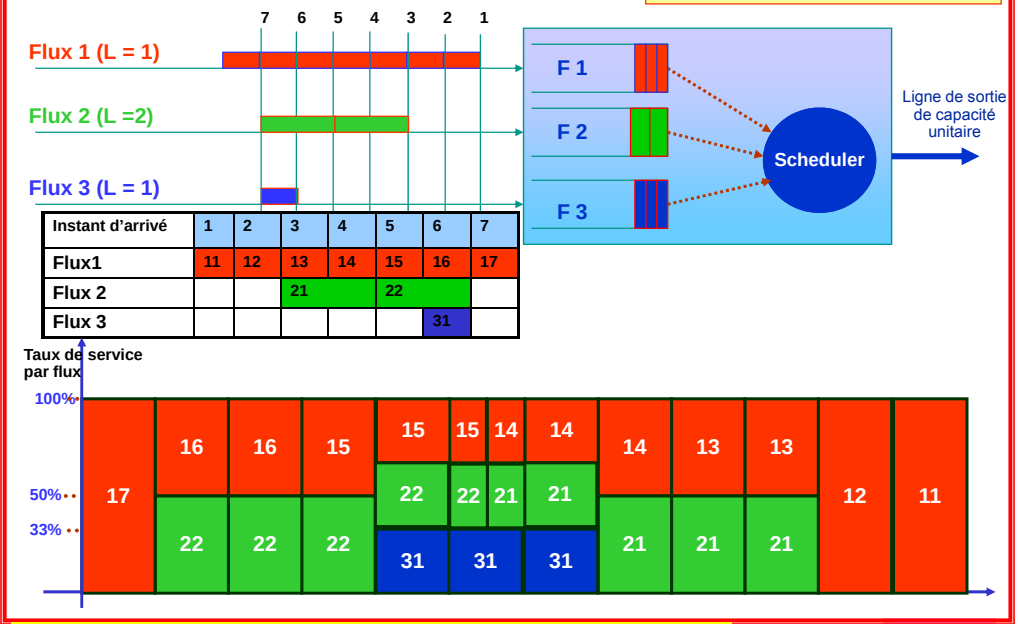
$$Débit[j](t) = \left(\frac{Débit\ du\ Lien\ de\ Sortie}{N_{Flux-Actif}(t)} \right)$$



L'ordonnanceur sert chaque queue comme si était un flux « liquide »

Processor Sharing (PS) « Partage du processeur » « Exemple»

$$Débit[j](t) = \left(\frac{Débit\ du\ Lien\ de\ Sortie}{N_{Flux-Actif}(t)} \right)$$



Instant d'arrivé	1	2	3	4	5	6	7
Flux1	11	12	13	14	15	16	17
Flux 2			21		22		
Flux 3						31	

Processor Sharing (PS)

- **Avantage :**
  - On peut montrer que si **on ne jette pas des paquets** et **sans faire aucune hypothèses** sur les source; simplement en construisant un **réseau de PS** , on aura un service qui est automatiquement équitable
    - ⇔on a une façon d'implémenter l'équité **Max-Min** sans dire à chaque source à quel débit, il doit travailler.
    - ⇔***La régulation se fait via le secheduler qu'on aura au niveau du réseau.***
    - ⇔Ceci ne dépend pas de contrôle de congestion dans TCP ou de son absence. Dépend seulement du fait qu'on jette pas de paquets dans les buffers.
    - ⇔Si on jette des paquets, on peut préserver l'équité **Max-Min** à condition que le **mécanisme qui jette des paquets soit équitable**
- **Inconvénient :** Solution " mathématique " idéale, non implantable dans la pratique
  - **Les** approximations de PS sont implantables
  - **Solutions :** différents approches sont possibles

Première implémentation de PS

« RR : Round Robin »

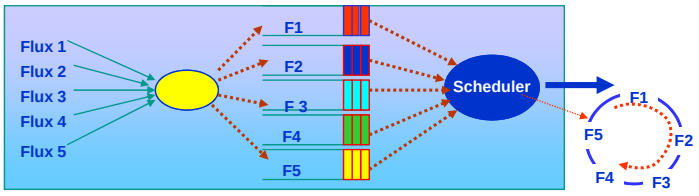
« A tour de rôle, paquet par paquet »

Nagle [RFC970].

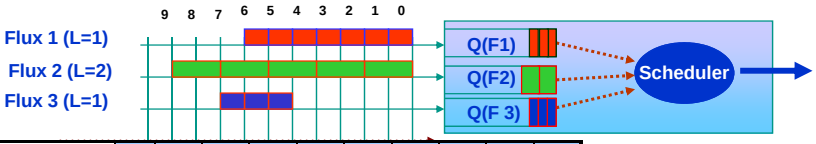
John B. Nagle. On packet switches with infinite storage. *IEEE Transactions on Communications*, COM-35(4):435--438, avril 1987.

Première implémentation de PS : Round Robin

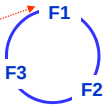
- **Principe de base :**
- L'idée de base est de construire **un agenda** qui nous indique à **quel moment on va servir un flux**.
- Cet agenda va **être un cercle** sur lequel, on va indiquer les instants auxquels il faut servir les queues 1, 2, …etc.
- Chaque fois, qu'on a à transmettre un paquet sur la liaison de sortie, le scheduler va regarder son agenda.
  - Si le pointeur se trouve devant F1, on va transmettre le paquet qui se trouve dans la queue F1. Pas de paquet dans la queue F1, on va regarder la queue F2, s'il y a un paquet dans F2, on le transmet, la transmission prend un certain temps.
  - Après la fin de transmission de paquet, on revient vers F2, s'il y a un autre paquet on le transmet, si non on passe au suivant … et ainsi de suite



Round Robin : Solution-1-



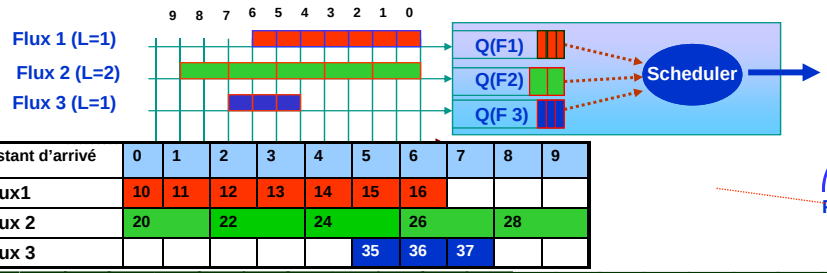
Instant d'arrivé	0	1	2	3	4	5	6	7	8	9
Flux1	10	11	12	13	14	15	16			
Flux 2	20		22		24		26		28	
Flux 3						35	36	37		



16	15	28	28	14	37	26	26	13	36	24	24	12	35	22	22	11	20	20	10

Temps	In F1	In F2	In F3	Q(F1)	Q(F2)	Q(F3)	Paquet Ordo.
0	P10[1]	P20[2]	-	P10	P20	-	F1:P10
1	P11[1]	-	-	P11	P20	-	F2:P20
2	P12[1]	P22[2]	-	P11,P12	P22	-	F2:P20 (cont)
3	P13[1]	-	-	P11,P12,P13	P22	-	F1:P11
4	P14[1]	P24[2]	-	P12,P13,P14	P22,P24	-	F2:P22
5	P15[1]	-	P35[1]	P12,P13,P14,P15	P24	P35	F2:P22 (cont)
6	P16[1]	P26[2]	P36[1]	P12,P13,P14,P15,P16	P24,P26	P35,P36	F3:P35
7	-	-	P37[1]	P12,P13,P14,P15,P16	P24,P26	P36,P37	F1:P12
8	-	P28[2]	-	P13,P14,P15,P16	P24,P26	P36,P37	F2:P24
9	-	-	-	P13,P14,P15,P16	P26,P28	P36,P37	F2:P24
10	-	P2A[2]	-	P13,P14,P15,P16	P26,P28	P36,P37	F3:P36

Round Robin : Solution-2-

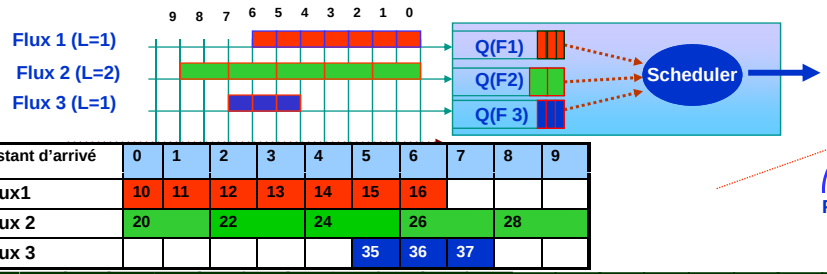


Instant d'arrivé	0	1	2	3	4	5	6	7	8	9
Flux1	10	11	12	13	14	15	16			
Flux 2	20		22		24		26		28	
Flux 3						35	36	37		

16	15	28	28	14	37	26	26	13	36	24	24	12	35	22	22	11	20	20	10

Temps	In F1	In F2	In F3	Q(F1)	Q(F2)	Q(F3)	Paquet Ordo.
0	P10[1]	P20[2]	-	P10	P20	-	F1:P10
1	P11[1]	-	-	P11	P20	-	F2:P20
2	P12[1]	P22[2]	-	P11,P12	P22	-	F2:P20 (cont)
3	P13[1]	-	-	P11,P12,P13	P22	-	F1:P11
4	P14[1]	P24[2]	-	P12,P13,P14	P22,P24	-	F2:P22
5	P15[1]	-	P35[1]	P12,P13,P14,P15	P24	P35	F2:P22 (cont)
6	P16[1]	P26[2]	P36[1]	P12,P13,P14,P15,P16	P24,P26	P35,P36	F3:P35
7	-	-	P37[1]	P12,P13,P14,P15,P16	P24,P26	P36,P37	F1:P12
8	-	P28[2]	-	P13,P14,P15,P16	P24,P26	P36,P37	F2:P24
9	-	-	-	P13,P14,P15,P16	P26,P28	P36,P37	F2:P24
10	-	P2A[2]	-	P13,P14,P15,P16	P26,P28	P36,P37	F3:P36

Round Robin : Solution-4-

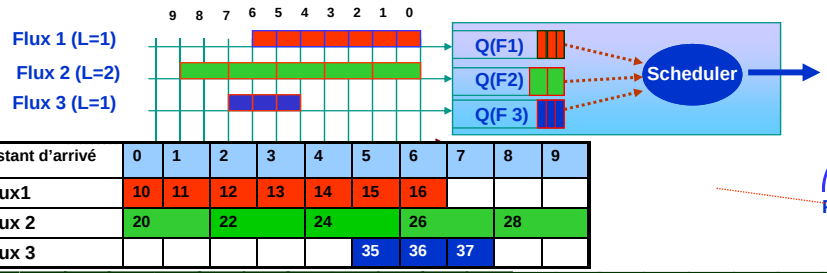


Instant d'arrivé	0	1	2	3	4	5	6	7	8	9
Flux1	10	11	12	13	14	15	16			
Flux 2	20		22		24		26		28	
Flux 3						35	36	37		

16	15	28	28	14	37	26	26	13	36	24	24	12	35	22	22	11	20	20	10

Temps	In F1	In F2	In F3	Q(F1)	Q(F2)	Q(F3)	Paquet Ordo.
0	P10[1]	P20[2]	-	P10	P20	-	F1:P10
1	P11[1]	-	-	P11	P20	-	F2:P20
2	P12[1]	P22[2]	-	P11,P12	P22	-	F2:P20 (cont)
3	P13[1]	-	-	P11,P12,P13	P22	-	F1:P11
4	P14[1]	P24[2]	-	P12,P13,P14	P22,P24	-	F2:P22
5	P15[1]	-	P35[1]	P12,P13,P14,P15	P24	P35	F2:P22 (cont)
6	P16[1]	P26[2]	P36[1]	P12,P13,P14,P15,P16	P24,P26	P35,P36	F3:P35
7	-	-	P37[1]	P12,P13,P14,P15,P16	P24,P26	P36,P37	F1:P12
8	-	P28[2]	-	P13,P14,P15,P16	P24,P26	P36,P37	F2:P24
9	-	-	-	P13,P14,P15,P16	P26,P28	P36,P37	F2:P24
10	-	P2A[2]	-	P13,P14,P15,P16	P26,P28	P36,P37	F3:P36

Round Robin : Solution-5-

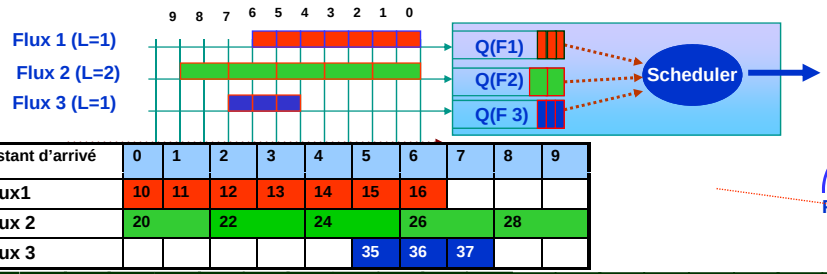


Instant d'arrivé	0	1	2	3	4	5	6	7	8	9
Flux1	10	11	12	13	14	15	16			
Flux 2	20		22		24		26		28	
Flux 3						35	36	37		

16	15	28	28	14	37	26	26	13	36	24	24	12	35	22	22	11	20	20	10

Temps	In F1	In F2	In F3	Q(F1)	Q(F2)	Q(F3)	Paquet Ordo.
0	P10[1]	P20[2]	-	P10	P20	-	F1:P10
1	P11[1]	-	-	P11	P20	-	F2:P20
2	P12[1]	P22[2]	-	P11,P12	P22	-	F2:P20 (cont)
3	P13[1]	-	-	P11,P12,P13	P22	-	F1:P11
4	P14[1]	P24[2]	-	P12,P13,P14	P22,P24	-	F2:P22
5	P15[1]	-	P35[1]	P12,P13,P14,P15	P24	P35	F2:P22 (cont)
6	P16[1]	P26[2]	P36[1]	P12,P13,P14,P15,P16	P24,P26	P35,P36	F3:P35
7	-	-	P37[1]	P12,P13,P14,P15,P16	P24,P26	P36,P37	F1:P12
8	-	P28[2]	-	P13,P14,P15,P16	P24,P26	P36,P37	F2:P24
9	-	-	-	P13,P14,P15,P16	P26,P28	P36,P37	F2:P24
10	-	P2A[2]	-	P13,P14,P15,P16	P26,P28	P36,P37	F3:P36

Round Robin : Solution-6-

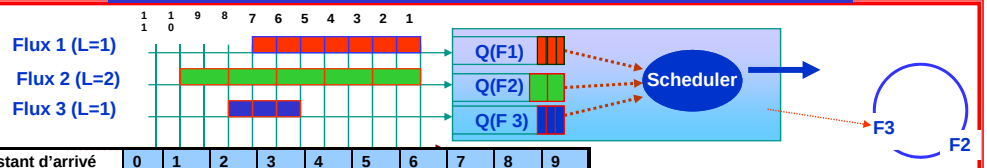


Instant d'arrivé	0	1	2	3	4	5	6	7	8	9
Flux1	10	11	12	13	14	15	16			
Flux 2	20		22		24		26		28	
Flux 3						35	36	37		

16	15	28	28	14	37	26	26	13	36	24	24	12	35	22	22	11	20	20	10

Temps	In F1	In F2	In F3	Q(F1)	Q(F2)	Q(F3)	Paquet Ordo.
0	P10[1]	P20[2]	-	P10	P20	-	F1:P10
1	P11[1]	-	-	P11	P20	-	F2:P20
2	P12[1]	P22[2]	-	P11,P12	P22	-	F2:P20 (cont)
3	P13[1]	-	-	P11,P12,P13	P22	-	F1:P11
4	P14[1]	P24[2]	-	P12,P13,P14	P22,P24	-	F2:P22
5	P15[1]	-	P35[1]	P12,P13,P14,P15	P24	P35	F2:P22 (cont)
6	P16[1]	P26[2]	P36[1]	P12,P13,P14,P15,P16	P24,P26	P35,P36	F3:P35
7	-	-	P37[1]	P12,P13,P14,P15,P16	P24,P26	P36,P37	F1:P12
8	-	P28[2]	-	P13,P14,P15,P16	P24,P26	P36,P37	F2:P24
9	-	-	-	P13,P14,P15,P16	P26,P28	P36,P37	F2:P24
10	-	P2A[2]	-	P13,P14,P15,P16	P26,P28	P36,P37	F3:P36

Round Robin : Solution-7-



Instant d'arrivé	0	1	2	3	4	5	6	7	8	9
Flux1	10	11	12	13	14	15	16			
Flux 2	20		22		24		26		28	
Flux 3						35	36	37		

⇔ On répartit bien le débit en terme de paquet, mais pas en terme d'octets

⇔ Flux vert : 50% de débit pour 5 paquets (30% en entrée)

16	15	28	28	14	37	26	26	13	36	24	24	12	35	22	22	11	20	20	10

Temps	In F1	In F2	In F3	Q(F1)	Q(F2)	Q(F3)	Paquet Ordo.
0	P10[1]	P20[2]	-	P10	P20	-	F1:P10
1	P11[1]	-	-	P11	P20	-	F2:P20
2	P12[1]	P22[2]	-	P11,P12	P22	-	F2:P20 (cont)
3	P13[1]	-	-	P11,P12,P13	P22	-	F1:P11
4	P14[1]	P24[2]	-	P12,P13,P14	P22,P24	-	F2:P22
5	P15[1]	-	P35[1]	P12,P13,P14,P15	P24	P35	F2:P22 (cont)
6	P16[1]	P26[2]	P36[1]	P12,P13,P14,P15,P16	P24,P26	P35,P36	F3:P35
7	-	-	P37[1]	P12,P13,P14,P15,P16	P24,P26	P36,P37	F1:P12
8	-	P28[2]	-	P13,P14,P15,P16	P24,P26	P36,P37	F2:P24
9	-	-	-	P13,P14,P15,P16	P26,P28	P36,P37	F2:P24
10	-	P2A[2]	-	P13,P14,P15,P16	P26,P28	P36,P37	F3:P36

Première implémentation de PS : Round Robin

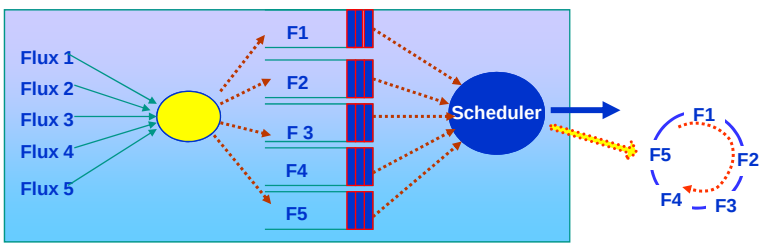
- **Avantages :**
  - Peut être facilement mis en application dans le matériel
  - Assure la protection pour le trafic **Best-Effort**
  - Fournit une distribution équitable de débit à condition qu'on **ait des paquets qui ont tous la même taille**
    - ⇔ il est très bien pour les réseaux ATM.
    - ⇔ moins bien pour les réseaux IP
  - **En effet, Dans un réseau IP** :
    - **50%** des paquets sont des paquets d'acquittement (**taille 40 octets**)
    - **20%** de paquets de taille **500 octets**
    - **30 %** de paquets de taille **1500 octets**
- **Inconvénients** :
  - **Non équitable** si la taille des paquets est variable

Première amélioration à RR

Deficit Round Robin : DRR

Deficit Round Robin -1-

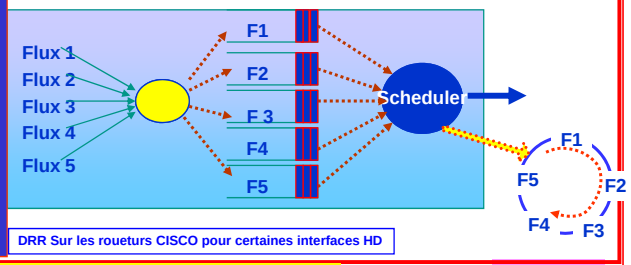
- **IDEES DE BASE :**
  1. On **conserve l'agenda** qui indique l'ordre dans lequel, il faut servir les différentes queues
  2. Au niveau de l'agenda, lorsque **le pointeur est en face** d'une **queue particulière**, on ne transmet pas **nécessairement** le paquet qui se trouve dans cette queue;
     - ✓ On attribue à chaque queue **un crédit** qui va **l'autoriser à transmettre**
     - ✓ une queue n'est pas autorisée à transmettre des paquets que s'elle a **accumulée un nombre de crédit** qui est suffisant pour couvrir la **taille de paquet** à transmettre)



Deficit Round Robin –2-

- **PRINCIPE :**
- Quand la **Queue i** est sélectionnée, on **augmente** le compteur (**Deficit**) associé à cette queue par une valeur **q**.
- Ensuite, on regarde si ***la taille du premier paquet*** de la ***queue [i]*** est plus grande que le **Deficit**.
  - ✓ **Si oui :** ⇔ la **Queue N° i** n'a pas accumulé suffisamment de crédit pour transmettre le paquet.
    Donc le paquet reste dans la queue.
  - ✓ **Si non** ⇔ il y a assez de crédit pour transmettre, on transmet le paquet sur la liaison de sortie et on **décrémente le Deficit** d'une valeur qui correspond à la **taille de paquet** qui a été transmis
  - ✓ **Si la queue est vide,** on réinitialise le **Deficit** pour éviter qu'une queue accumule très vite des Dificit inutilement

```
// Associer un compteur D[i] à chaque queue logique
// Soit L, la taille du premier paquet dans la queue
// Chaque fois qu'une queue logique Q(i) est visitée
(⇔ pas forcément servi)
Si (L ≥ D[i])
   le paquet reste dans la queue
Sinon {
   Le paquet est transmis sur le lien de sortie
   D[i] = D[i] - L
   Si ( la file est vide après transmission) alors
        D[i] = 0
}
```



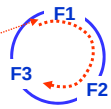
DRR Sur les roueturs CISCO pour certaines interfaces HD

Deficit Round Robin : Solution

Exemple avec q = 1

Instant d'arrivé	0	1	2	3	4	5	6	7	8	9
Flux1	10	11	12	13	14	15	16			
Flux 2	20		22		24		26		28	
Flux 3						35	36	37		

Agenda sur

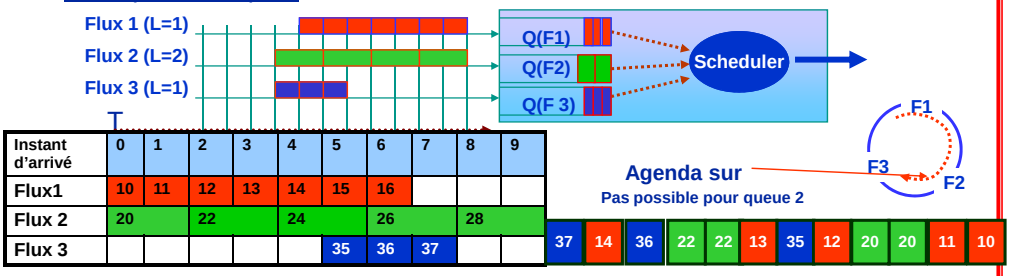


37	14	36	22	22	13	35	12	20	20	11	10

T	Inc.	D[1]	D[2]	D[3]	Q(F1)	Q(F2)	Q(F3)	paquet ordonnancé
0	F1	0+1-1	0	0	P10	P20	-	F1:P10
1	F2,F1	0+1-1	0+1	0	P11	P20	-	F1:P11
2	F2	0	1+1-2	0	P12	P22	-	F2:P20
3	-	0	0	0	P13	P22	-	F2:P20(cont)
4	F1	0+1-1	0	0	P14	P22,P24	-	F1:P12
5	F2,F3	0	0+1	0+1-1	P14,P15	P22,P24	P35	F3:P35
6	F1	0+1-1	0	0	P14,P15,P16	P22,P24,P26	P36	F1:P13
7	F2	0	1+1-2		P15,P16	P22,P24,P26	P36,P37	F2:P22
8	-	0	0	0	P15,P16	P24,P26	P36,P37	F2:P22(cont)
9	F3	0	0	0+1-1	P15,P16	P24,P26	P36,P37	F3:P36
10	F1	0+1-1	0	0	P15,P16	P24,P26	P37	F1:P14
11	F2,F3	0	0+1	0+1-1	P15,P16	P24,P26	P37	F3:P37

Deficit Round Robin : Solution

Exemple avec q = 1

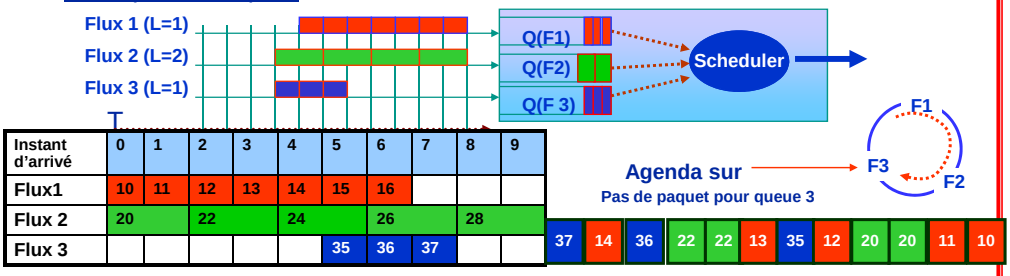


Instant d'arrivé	0	1	2	3	4	5	6	7	8	9
Flux1	10	11	12	13	14	15	16			
Flux 2	20		22		24		26		28	
Flux 3						35	36	37		

T	Inc.	D[1]	D[2]	D[3]	Q(F1)	Q(F2)	Q(F3)	paquet ordonnancé
0	F1	0+1-1	0	0	P10	P20	-	F1:P10
1	F2,F1	0+1-1	0+1	0	P11	P20	-	F1:P11
2	F2	0	1+1-2	0	P12	P22	-	F2:P20
3	-	0	0	0	P13	P22	-	F2:P20(cont)
4	F1	0+1-1	0	0	P14	P22,P24	-	F1:P12
5	F2,F3	0	0+1	0+1-1	P14,P15	P22,P24	P35	F3:P35
6	F1	0+1-1	0	0	P14,P15,P16	P22,P24,P26	P36	F1:P13
7	F2	0	1+1-2		P15,P16	P22,P24,P26	P36,P37	F2:P22
8	-	0	0	0	P15,P16	P24,P26	P36,P37	F2:P22(cont)
9	F3	0	0	0+1-1	P15,P16	P24,P26	P36,P37	F3:P36
10	F1	0+1-1	0	0	P15,P16	P24,P26	P37	F1:P14
11	F2,F3	0	0+1	0+1-1	P15,P16	P24,P26	P37	F3:P37

Deficit Round Robin : Solution

Exemple avec q = 1

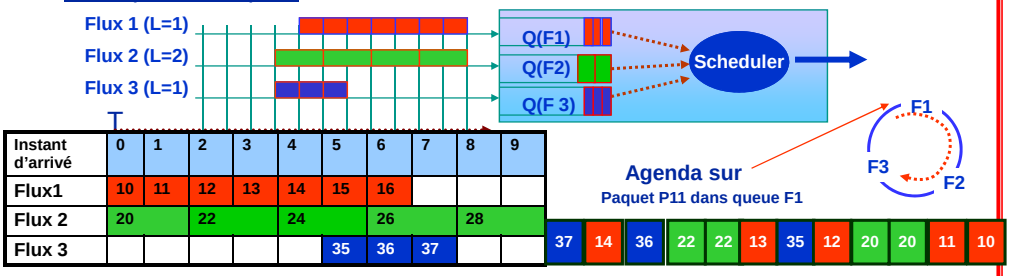


Instant d'arrivé	0	1	2	3	4	5	6	7	8	9
Flux1	10	11	12	13	14	15	16			
Flux 2	20		22		24		26		28	
Flux 3						35	36	37		

T	Inc.	D[1]	D[2]	D[3]	Q(F1)	Q(F2)	Q(F3)	paquet ordonnancé
0	F1	0+1-1	0	0	P10	P20	-	F1:P10
1	F2,F1	0+1-1	0+1	0	P11	P20	-	F1:P11
2	F2	0	1+1-2	0	P12	P22	-	F2:P20
3	-	0	0	0	P13	P22	-	F2:P20(cont)
4	F1	0+1-1	0	0	P14	P22,P24	-	F1:P12
5	F2,F3	0	0+1	0+1-1	P14,P15	P22,P24	P35	F3:P35
6	F1	0+1-1	0	0	P14,P15,P16	P22,P24,P26	P36	F1:P13
7	F2	0	1+1-2		P15,P16	P22,P24,P26	P36,P37	F2:P22
8	-	0	0	0	P15,P16	P24,P26	P36,P37	F2:P22(cont)
9	F3	0	0	0+1-1	P15,P16	P24,P26	P36,P37	F3:P36
10	F1	0+1-1	0	0	P15,P16	P24,P26	P37	F1:P14
11	F2,F3	0	0+1	0+1-1	P15,P16	P24,P26	P37	F3:P37

Deficit Round Robin : Solution

Exemple avec q = 1



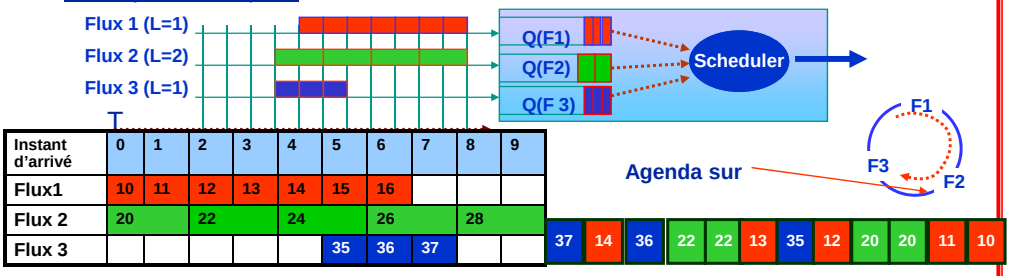
Instant d'arrivé	0	1	2	3	4	5	6	7	8	9
Flux1	10	11	12	13	14	15	16			
Flux 2	20		22		24		26		28	
Flux 3						35	36	37		

37 14 36 22 22 13 35 12 20 20 11 10

T	Inc.	D[1]	D[2]	D[3]	Q(F1)	Q(F2)	Q(F3)	paquet ordonnancé
0	F1	0+1-1	0	0	P10	P20	-	F1:P10
1	F2,F1	0+1-1	0+1	0	P11	P20	-	F1:P11
2	F2	0	1+1-2	0	P12	P22	-	F2:P20
3	-	0	0	0	P13	P22	-	F2:P20(cont)
4	F1	0+1-1	0	0	P14	P22,P24	-	F1:P12
5	F2,F3	0	0+1	0+1-1	P14,P15	P22,P24	P35	F3:P35
6	F1	0+1-1	0	0	P14,P15,P16	P22,P24,P26	P36	F1:P13
7	F2	0	1+1-2		P15,P16	P22,P24,P26	P36,P37	F2:P22
8	-	0	0	0	P15,P16	P24,P26	P36,P37	F2:P22(cont)
9	F3	0	0	0+1-1	P15,P16	P24,P26	P36,P37	F3:P36
10	F1	0+1-1	0	0	P15,P16	P24,P26	P37	F1:P14
11	F2,F3	0	0+1	0+1-1	P15,P16	P24,P26	P37	F3:P37

Deficit Round Robin : Solution

Exemple avec q = 1



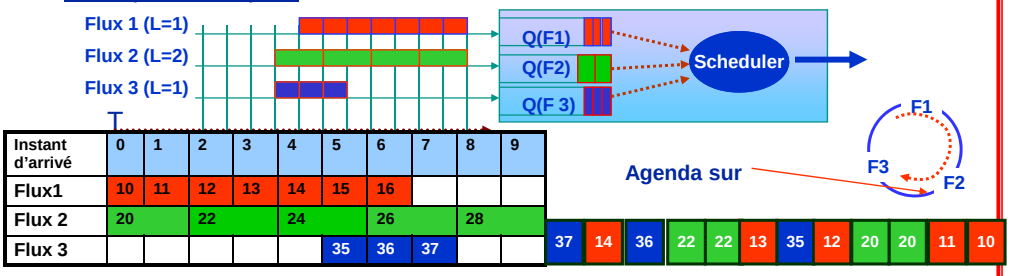
Instant d'arrivé	0	1	2	3	4	5	6	7	8	9
Flux1	10	11	12	13	14	15	16			
Flux 2	20		22		24		26		28	
Flux 3						35	36	37		

37 14 36 22 22 13 35 12 20 20 11 10

T	Inc.	D[1]	D[2]	D[3]	Q(F1)	Q(F2)	Q(F3)	paquet ordonnancé
0	F1	0+1-1	0	0	P10	P20	-	F1:P10
1	F2,F1	0+1-1	0+1	0	P11	P20	-	F1:P11
2	F2	0	1+1-2	0	P12	P22	-	F2:P20
3	-	0	0	0	P12,P13	P22	-	F2:P20(cont)
4	F1	0+1-1	0	0	P12,P13,P14	P22,P24	-	F1:P12
5	F2,F3	0	0+1	0+1-1	P14,P15	P22,P24	P35	F3:P35
6	F1	0+1-1	0	0	P14,P15,P16	P22,P24,P26	P36	F1:P13
7	F2	0	1+1-2		P15,P16	P22,P24,P26	P36,P37	F2:P22
8	-	0	0	0	P15,P16	P24,P26	P36,P37	F2:P22(cont)
9	F3	0	0	0+1-1	P15,P16	P24,P26	P36,P37	F3:P36
10	F1	0+1-1	0	0	P15,P16	P24,P26	P37	F1:P14
11	F2,F3	0	0+1	0+1-1	P15,P16	P24,P26	P37	F3:P37

Deficit Round Robin : Solution

Exemple avec q = 1



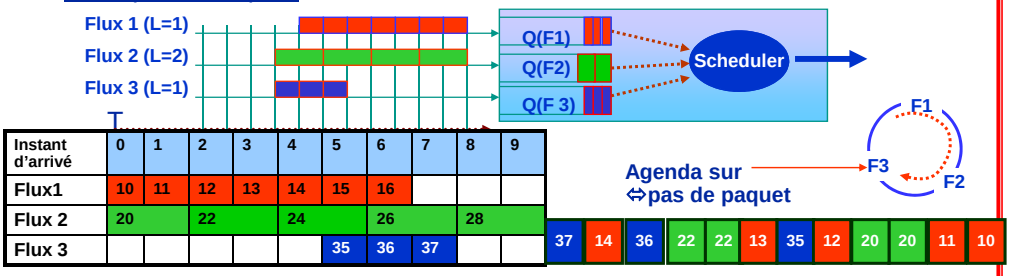
Instant d'arrivé	0	1	2	3	4	5	6	7	8	9
Flux1	10	11	12	13	14	15	16			
Flux 2	20		22		24		26		28	
Flux 3						35	36	37		

37 14 36 22 22 13 35 12 20 20 11 10

T	Inc.	D[1]	D[2]	D[3]	Q(F1)	Q(F2)	Q(F3)	paquet ordonnancé
0	F1	0+1-1	0	0	P10	P20	-	F1:P10
1	F2,F1	0+1-1	0+1	0	P11	P20	-	F1:P11
2	F2	0	1+1-2	0	P12	P22	-	F2:P20
3	-	0	0	0	P12,P13	P22	-	F2:P20(cont)
4	F1	0+1-1	0	0	P12,P13,P14	P22,P24	-	F1:P12
5	F2,F3	0	0+1	0+1-1	P14,P15	P22,P24	P35	F3:P35
6	F1	0+1-1	0	0	P14,P15,P16	P22,P24,P26	P36	F1:P13
7	F2	0	1+1-2		P15,P16	P22,P24,P26	P36,P37	F2:P22
8	-	0	0	0	P15,P16	P24,P26	P36,P37	F2:P22(cont)
9	F3	0	0	0+1-1	P15,P16	P24,P26	P36,P37	F3:P36
10	F1	0+1-1	0	0	P15,P16	P24,P26	P37	F1:P14
11	F2,F3	0	0+1	0+1-1	P15,P16	P24,P26	P37	F3:P37

Deficit Round Robin : Solution

Exemple avec q = 1



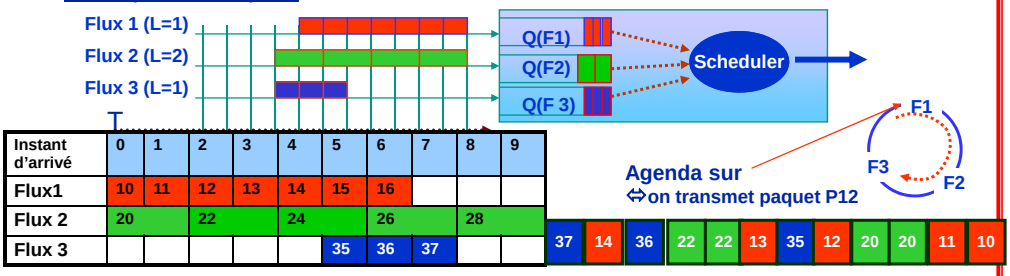
Instant d'arrivé	0	1	2	3	4	5	6	7	8	9
Flux1	10	11	12	13	14	15	16			
Flux 2	20		22		24		26		28	
Flux 3						35	36	37		

37 14 36 22 22 13 35 12 20 20 11 10

T	Inc.	D[1]	D[2]	D[3]	Q(F1)	Q(F2)	Q(F3)	paquet ordonnancé
0	F1	0+1-1	0	0	P10	P20	-	F1:P10
1	F2,F1	0+1-1	0+1	0	P11	P20	-	F1:P11
2	F2	0	1+1-2	0	P12	P22	-	F2:P20
3	-	0	0	0	P12,P13	P22	-	F2:P20(cont)
4	F1	0+1-1	0	0	P12,P13,P14	P22,P24	-	F1:P12
5	F2,F3	0	0+1	0+1-1	P14,P15	P22,P24	P35	F3:P35
6	F1	0+1-1	0	0	P14,P15,P16	P22,P24,P26	P36	F1:P13
7	F2	0	1+1-2		P15,P16	P22,P24,P26	P36,P37	F2:P22
8	-	0	0	0	P15,P16	P24,P26	P36,P37	F2:P22(cont)
9	F3	0	0	0+1-1	P15,P16	P24,P26	P36,P37	F3:P36
10	F1	0+1-1	0	0	P15,P16	P24,P26	P37	F1:P14
11	F2,F3	0	0+1	0+1-1	P15,P16	P24,P26	P37	F3:P37

Deficit Round Robin : Solution

Exemple avec q = 1



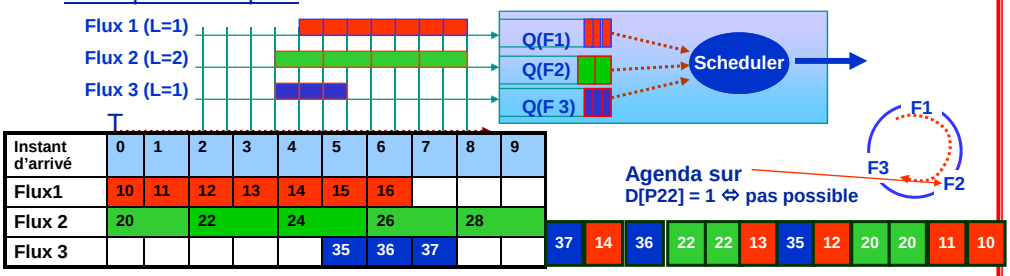
Instant d'arrivé	0	1	2	3	4	5	6	7	8	9
Flux1	10	11	12	13	14	15	16			
Flux 2	20		22		24		26		28	
Flux 3						35	36	37		

37 14 36 22 22 13 35 12 20 20 11 10

T	Inc.	D[1]	D[2]	D[3]	Q(F1)	Q(F2)	Q(F3)	paquet ordonnancé
0	F1	0+1-1	0	0	P10	P20	-	F1:P10
1	F2,F1	0+1-1	0+1	0	P11	P20	-	F1:P11
2	F2	0	1+1-2	0	P12	P22	-	F2:P20
3	-	0	0	0	P12,P13	P22	-	F2:P20(cont)
4	F1	0+1-1	0	0	P12,P13,P14	P22,P24	-	F1:P12
5	F2,F3	0	0+1	0+1-1	P14,P15	P22,P24	P35	F3:P35
6	F1	0+1-1	0	0	P14,P15,P16	P22,P24,P26	P36	F1:P13
7	F2	0	1+1-2		P15,P16	P22,P24,P26	P36,P37	F2:P22
8	-	0	0	0	P15,P16	P24,P26	P36,P37	F2:P22(cont)
9	F3	0	0	0+1-1	P15,P16	P24,P26	P36,P37	F3:P36
10	F1	0+1-1	0	0	P15,P16	P24,P26	P37	F1:P14
11	F2,F3	0	0+1	0+1-1	P15,P16	P24,P26	P37	F3:P37

Deficit Round Robin : Solution

Exemple avec q = 1



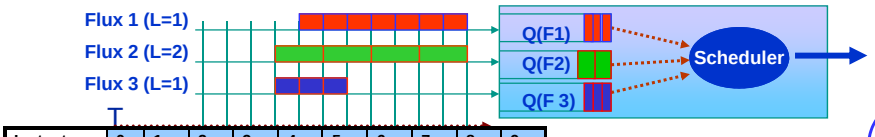
Instant d'arrivé	0	1	2	3	4	5	6	7	8	9
Flux1	10	11	12	13	14	15	16			
Flux 2	20		22		24		26		28	
Flux 3						35	36	37		

37 14 36 22 22 13 35 12 20 20 11 10

T	Inc.	D[1]	D[2]	D[3]	Q(F1)	Q(F2)	Q(F3)	paquet ordonnancé
0	F1	0+1-1	0	0	P10	P20	-	F1:P10
1	F2,F1	0+1-1	0+1	0	P11	P20	-	F1:P11
2	F2	0	1+1-2	0	P12	P22	-	F2:P20
3	-	0	0	0	P12,P13	P22	-	F2:P20(cont)
4	F1	0+1-1	0	0	P12,P13,P14	P22,P24	-	F1:P12
5	F2,F3	0	0+1	0	P14,P15	P22,P24	P35	F3:P35
6	F1	0+1-1	0	0	P14,P15,P16	P22,P24,P26	P36	F1:P13
7	F2	0	1+1-2		P15,P16	P22,P24,P26	P36,P37	F2:P22
8	-	0	0	0	P15,P16	P24,P26	P36,P37	F2:P22(cont)
9	F3	0	0	0+1-1	P15,P16	P24,P26	P36,P37	F3:P36
10	F1	0+1-1	0	0	P15,P16	P24,P26	P37	F1:P14
11	F2,F3	0	0+1	0+1-1	P15,P16	P24,P26	P37	F3:P37

Deficit Round Robin : Solution

Exemple avec q = 1



Instant d'arrivé	0	1	2	3	4	5	6	7	8	9
Flux1	10	11	12	13	14	15	16			
Flux 2	20		22		24		26		28	
Flux 3						35	36	37		

Agenda sur
D[P35] = 1 ⇔ on transmet P35

37 14 36 22 22 13 35 12 20 20 11 10

T	Inc.	D[1]	D[2]	D[3]	Q(F1)	Q(F2)	Q(F3)	paquet ordonnancé
0	F1	0+1-1	0	0	P10	P20	-	F1:P10
1	F2,F1	0+1-1	0+1	0	P11	P20	-	F1:P11
2	F2	0	1+1-2	0	P12	P22	-	F2:P20
3	-	0	0	0	P12,P13	P22	-	F2:P20(cont)
4	F1	0+1-1	0	0	P12,P13,P14	P22,P24	-	F1:P12
5	F2,F3	0	0+1	0+1-1	P14,P15	P22,P24	P35	F3:P35
6	F1	0+1-1	0	0	P14,P15,P16	P22,P24,P26	P36	F1:P13
7	F2	0	1+1-2		P15,P16	P22,P24,P26	P36,P37	F2:P22
8	-	0	0	0	P15,P16	P24,P26	P36,P37	F2:P22(cont)
9	F3	0	0	0+1-1	P15,P16	P24,P26	P36,P37	F3:P36
10	F1	0+1-1	0	0	P15,P16	P24,P26	P37	F1:P14
11	F2,F3	0	0+1	0+1-1	P15,P16	P24,P26	P37	F3:P37

Dernier Problème

Quel paquet jeter « supprimer »?

Quel paquets jeter ? -1-

- Il faut aussi qu'on est un mécanisme en amont qui nous permet de jeter les paquets de façon équitable

- Comment jeter les paquets de manière équitable lorsqu'on utilise un ordonnanceur Best-Effort ?

- Principe :

  - Jeter les paquets qui sont les **plus responsables de la congestion** ⇔ des paquets qui correspondent à des connexions TCP qui envoient **beaucoup plus de paquets** que les **ressources** dont elles **disposent sur la liaison de sortie**

  - Si on a une connexion qui a un **Débit > au Débit qui lui a été alloué** sur la liaison de sortie : son occupation de buffer **va augmenter**

Quel paquets jeter ? -2-

- Solution :

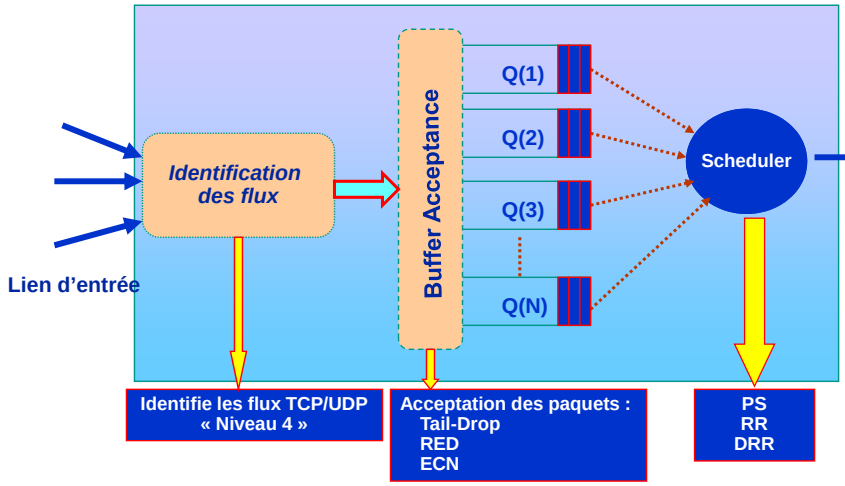
  - **Donc pour jeter les paquets de façon équitable,** il suffit de regarder le buffer **qui a le plus de paquets**, car c'est celle qui est la plus responsable de la congestion.

- Conclusion

  - **Donc, grâce au fait qu'on a un scheduller et une queue pour chaque connexion, on peut facilement identifier quelle connexion est responsable de la congestion**

  - **Jeter le (s) paquet(s) de la plus longue file d'attente**
    1. **A la queue de la file d'attente**
    2. **A la tête de la file d'attente**
    3. **De manière aléatoire**

Bilan : « Routeur Best-Effort – Version améliorée »



Contrôle de trafic et QoS dans les réseaux IP

1. Applications
2. Mécanismes de contrôle de trafic au niveau paquet
   - ⇨ Service Best-effort
   - ⇨ Service dé débit maximum garantie
   - ⇨ Service dé débit minimum garantie
   - ⇨ Garanties de délai
3. Services Normalisés
4. Contrôle de trafic d'Intra-domaine
5. Contrôle de trafic d'Inter-domaine
6. Études de cas

Quel type de garanties? -1-

- Trois types de garanties - rappels :
  1. Le Best effort (aucune garantie de débit « bande passante »)
     - Approprié aux applications classiques, non critiques, sporadiques (élastiques)
  2. Services avec Débit Maximum Garantie (Maximum Guaranteed Bandwidth)
     - Une part des ressources du réseau (débit) **est réservée** à un flux donné
     - Ce flux ne peut pas dépasser débit qui lui est allouée
     - Ce type de services est approprié aux applications du type **streaming non-adaptatives**, ou bien pour **contrôler le réseau** pour éviter que les utilisateurs ne consomment trop de ressources à l'intérieur du réseau

Quel type de garanties? -2-

3. Services avec Débit minimum garantie (Minimum Guaranteed bandwidth)
   - A tout moment, permettre à une source d'avoir la garantie qu'elle aura**, au moins, un minimum de ressources à sa disposition dans le réseau**
   - Si la source souhaite envoyer plus vite que son **débit min. garantie**, elle peut le faire, mais ça sera à son risque et ce que la source transmettra en surplus de son débit min garantie, sera en **Best-Effot** :
     - Si le réseau n'est pas congestionné : **ça passera**
     - Si non, il faut que l'application **puisse s'adapter**
   - Approprié aux **applications critiques élastiques** et du type streaming adaptatif

Quel type de garanties? - 3-

- On aura aussi à prendre en compte les ***Garantie de délai*** et de ***variation de délai (jitter)***

1. ***Pour les Flux Best effort :***
   - pas de garantie de délai et de jitter
2. ***Pour les flux à Débit maximum garantie :***
   - On pourra rajouter une Garantie maximum sur les délais (⇔ borne max)
     - Par exemple pour la voix interactive
   - **On peut Garantir un maximum sur la variabilité de délai** « de jitter », mais pas de garantie sur la variation de délai (car ceci n'a pas de signification ou moins dans un réseau IP)
3. *Pour les flux à **Débit minimum garantie** :*
   - **On pourra fournir une Garantie de délai maximale**
     - **Par exemple pour les application du type streaming adaptatives**
   - **Sur la variabilité de délai** : on peut rien faire « ceci n'a pas beaucoup de signification » à partir de moment ou on autorise l'application à envoyer un surplus de débit en **Best-Effort**

Services avec garantie de Débit maximum

Services Best effot vs services garanties

1. « Provision » des services Best effort ?
   - Peuvent être faits en considérant que tous les paquets d'IP veulent recevoir exactement le même service (on doit s'arranger pour fournir une équité Max-Min)
     - **(⇔ tous les paquets IP sont égaux)**

2. Provision des services garanties ?
   - Tous les paquets IP **ne sont plus égaux** : Donc, on pourra pas traiter ces paquets de la même façon.
   - Donc, il faut qu'on ait, au moins, dans certain endroits du réseau des équipements qu'ils soient capable de **différencier** les **paquets garanties** des **paquets non garanties**
     - **éventuellement quels sont les types de garanties associés à chaque type de paquets**
   - En plus, cette notion « garantie de débit » ne peut plus être appliquée à un paquet **mais à un flux**
     - **On mesure pas le débit d'un paquet, mais le débit d'une suite de paquets**

⇔ *la garantie de débit va s'appliquer à flux de paquets*

Best effot vs services garanties

Pour réaliser ça, deux problèmes à résoudre :

1. **Associer les paquets IP à des flux (pour lesquels on veut fournir des garanties)**
2. **Fournir des garanties à des flux**

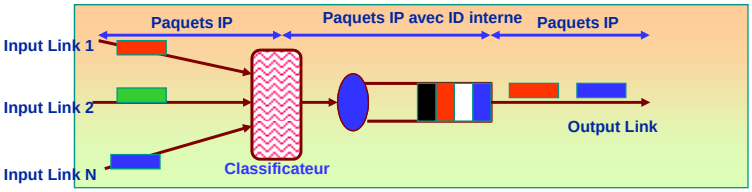
Qu'est-ce qu'un flux ?

- Définition générale :
  - Un flux est une séquence de paquets possédant une **caractéristique commune :**
    - Une caractéristique peut être basée sur n'importe quel champ des paquets
    - Un flux a une existence; un début de flux et une fin (⇔ **pour une période**)

  - Au niveau du réseau, on va retrouver des flux, quasiment à chaque niveau de modèle OSI.
  - Un flux de niveau N est une séquence de paquets possédant une caractéristique commune de niveau N
    - ***Exemple de niveau 2***
      - Identificateur de connexion **(DLCI)** des trames frame-relay
    - ***Exemple de niveau 3***
      - Adresses source et destination des **paquets IP**
    - ***Exemple de niveau 4***
      - Ports source et destination des **TPDU (TCP ou UDP)**

routeur simple - v2 -

- **Pour traiter ces flux, on va devoir améliorer notre routeur (⇔ ajouter un mécanisme de classification).**
- **Objectifs de la fonction de classification dans les routeurs**
- **Pour chaque paquet entrant**
  1. **Déterminer son flux appartenance (identification du flux)**
     - Peut nécessiter des **opérations complexes**
     - Parfois impossible en cas **d'utilisation de tunnels** (identification à faire avant tout cryptage)
  2. **Enregistrer cette information (marquage du flux)**
     - **Après identification du flux :** ce que on va faire est de **marquer le paquet avec un identificateur interne du flux** de tel façon pour que les mécanismes qui se trouvent en aval du routeur puissent prendre en compte le **flux auquel un paquet appartient sans reprendre la classification.**
     - Ceci sera un identificateur interne (⇔ ne sera pas exporté sur la liaison de sortie)



Identification des flux

Flux du niveau deux

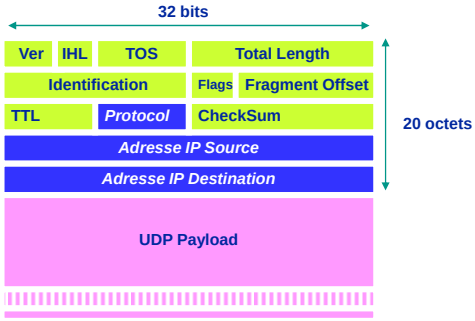
- Problème : Comment faire la classification ?
- Réponse : ceci dépend de quel type de flux on va utiliser

- Certains technologies de la couche deux fournit des mécanismes spécifiques pour l'identification des flux

1. **ATM :**
   VCI/VPI
2. **Frame Relay**
   DLCI
3. **802.x LANs**
   VLANs, 802.1q, 802.1d, 802.1p

Flux du niveau trois –1-

- Identification des flux du niveau trois (couche réseau):
  - **En intervenant au niveau des adresses IP de source et de destination avec ou sans Netmasks associés**
    - Par exemple tout le trafic de **138.48.0.0/16** (venant de tel préfix ou vers tel préfix)
  - **En intervenant au niveau des protocoles de routage :**
    - Tout le trafic d'IP qui a le même chemin ou BGP Next Hop
    - Cette classification exige de consulter la table de routage par le classificateur

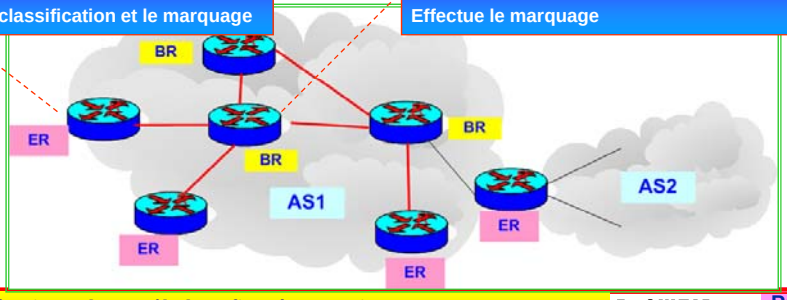


Flux du niveau trois - 2 -

- **A l'échelle d'un grand réseau, demander à chaque routeur de faire la classification est une opération coûteuse en traitement**
- **Solution alternative** :
  - Effectuer **la classification** dans les routeurs qui se trouvent à la frontière du SA **(Edge Routers)**
  - Une fois qu'ils ont fait la classification, qu'ils **marquent explicitement le flux auquel**, le paquet appartient de façon à ce que à l'intérieur **du réseau de Backbone** on puisse utiliser cette classification pour prendre des décisions sur le flux auquel le paquet appartienne sans **devoir refaire une classification complexe à l'intérieur des routeurs de backbone**

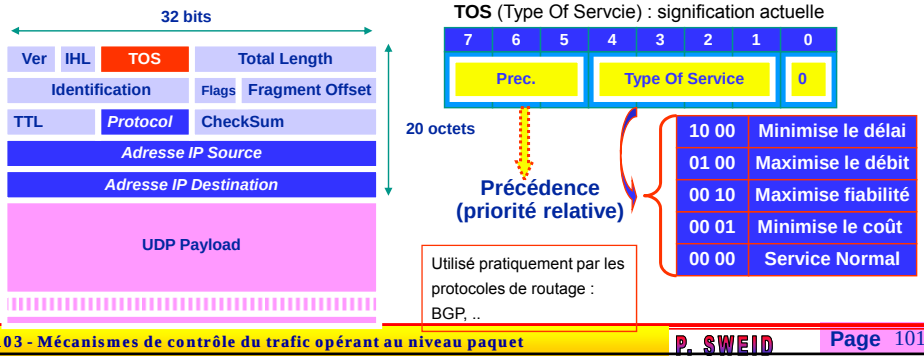
ER : Routeur de bordure « Edge Router »
Effectue la classification et le marquage

BR : Routeur de Backbon « Backbone Router »
Effectue le marquage



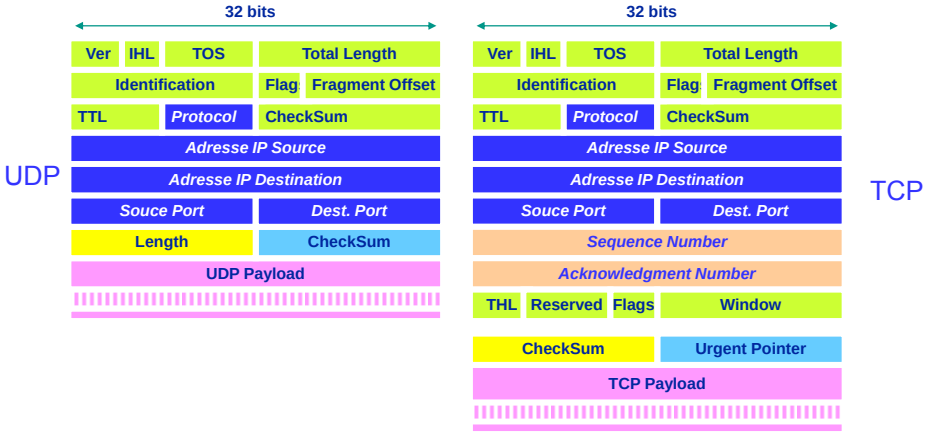
Marquage des paquets IP

- **Comment effectuer explicitement le marquage d'un paquet IP ?**
  - **Dérober un champ de l'en-tête d'IP :**
    - **TOS (Type Of Servcie) :** Type d'octet de service
    - Définit l'importance relative du paquet d'IP et le type de service requis pour ce paquet
  - **Situation actuelle :**
    - La définition de l'octet de TOS a changé plusieurs fois
    - Le champ « précédence » priorité est employée dans quelques réseaux
    - Le champ de TOS est rarement employé
  - **Employer l'octet de TOS pour le marquage :**
    - **Avantage :** facile à implémenter
    - **Inconvénient :** nombre limité de flux marqués



Flux du niveau quatre

- Flux du niveau quatre identifié par le quintuple :
  1. Adresse IP source
  2. Adresse IP de destination
  3. Type de Protocole,
  4. N° du port source,
  5. N° du port de destination



Identification des applications

- Solution plus « simple »:
  - Utiliser le numéros de ports **TCP/UDP** pour identifier les **applications le plus importantes**

Application	Protocole de Transport	Port number
DHCP/Bootp	UDP	67, 68
DNS	UDP	53
HTTP	TCP	80
HTTP	TCP	443
IMAP	TCP/UDP	143, 220
LDAP	TCP/UDP	389
MS-SQL	TCP	1433
NetBIOS	TCP	137, 139
NFS	TCP/UDP	2049
MNTP	TCP/UDP	119
Lotus Notes	TCP/UDP	352
POP	TCP/UDP	109/110
SMTP	TCP	25
SNMP	TCP/UDP	161, 162
SSH	TCP	22
Syslog	UDP	14
Telnet	TCP	23
X Windows	TCP	6000-6003

Identification des applications - 2 -

- Le problème :
  - *Pas toutes les applications utilisent les ports dits « well-know »*
  - *Exemples*
    - Ftp
      - le serveur et le client peuvent négocier des ports différents des ports 20/21 pour effectuer le transfert des fichiers (mode passif)
    - RealAudio/NetShow (microsoft)
      - Dans Le protocole **RTSP, le port 554** est utilisé pour négocier le contenu à échanger ainsi que les ports UDP/TCP à utiliser
    - SunRPC
      - Les services RPC utilisent un numéro de port négocié : portmap/rpcbind (port 111) est utilisé pour localiser le service « sur un serveur »
    - Napster/Gnutella et d'autres nouvelles applications
      - utilisent rarement des ports connus **« well-known »**
      - Caza ; change régulièrement le n° de port pour éviter de l'identifier

Identification des applications - 3 -

- Comment classifier des paquets des applications comme ftp/RealAudio/RPC?
- Principe de la solution : Examinez le contenu des paquets de de contrôle de telles applications
  - Commandes ftp, RTSP, portmap...
  - Utiliser ces informations obtenues pour **effectuer une classification du niveau 4**
- Inconvénients :
  - Une solution consommatrice du temps CPU (il faut une application qui regarde pour chaque application les ports.
    - *Solution est aussi propriétaire car dépend de l'implémentation*
  - Des fonctions « handler » spécifiques doivent être écrites et utilisées pour capturer et traiter les informations véhiculées par ces protocoles de contrôle.

Identification des applications – D'autres Limitations -

- Certains applications sont plus difficilement identifiables et non connues
  - Il suffit d'analyser le trafic internent pour constater cette situation
- Déploiement des tunnels chiffrés (IPSEC, L2TP, PPTP) qui cachent les en-têtes TCP et le UDP aux routeurs intermédiaires
  - Si un traitement particulier est nécessaire à l'intérieur du réseau, les paquets devraient être marqués au niveau de la couche 3 avant qu'ils soient chiffrés
  - Le déploiement de chiffrage simultanément avec le contrôle de trafic devrait être effectué de manière prudente

Flux du niveau 7

- **Exemple d'un flux du niveau 7 :**
  - Des documents du type text/html
  - Tout types d'image …etc.
  - Très coûteux car il faut regarder le contenu du paquet
- **Exemple d'une page WWW**

```
<!DOCTYPE HTML PUBLIC " //W3C//DTD HTML 3.2//EN">
<HTML>
<BODY BGCOLOR="#ffffff" BACKGROUND="/icons/back.gif">
This is a simple html page.<P>
<IMG SRC="left.gif" ALT="Left">
<IMG SRC="right.gif" ALT="Right">
</BODY>
</HTML>
```

Extraction avec HTTP 1.0 :

Conn. TCP . pour html
Conn. TCP pour background (back.gif)
Conn. TCP pour left.gif
Conn. TCP pour right.gif

Extraction avec HTTP 1.1 :

Tout, dans une seule connexion TCP (html page, back.gif, left.gif , right.gif)

Classification , à Quel niveau ? -1-

- Dans quelle couche, effectuer la classification ?
- la classification layer-7 est très coûteux
  - ✓ Exige l'examen des **en-têtes** et du **contenu de paquet**
    - Ne marche pas à haut débit et au niveau de Backbone
  - ✓ Peut être probablement employée par des **équipements spécialisé de contrôle de trafic à 2 Mbits/s ou à 10 Mbits/s ou bien dans des équipements comme les Firwals**
  - ✓ Il y a des équipement qui permet d'effectuer des classification au niveau Web comme par exemple classifier les sessions HTTP qui contiennent des images **GIF(**pour un fournisseur de contenu que veut favoriser l'envoie des contenus avec des images **GIF**). Pour un utilisateur Internet ; qui veut par contre favoriser la réception des pages HTML

Classification , à Quel niveau ? -2-

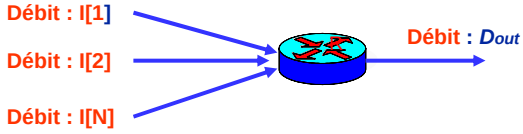
- Classification du niveau 3 contre la classification du niveau 4
  - Aucun vrai consensus aujourd'hui
  - Certains pensent que la classification du **niveau 4** peut être effectuée par chaque routeur, même dans les routeurs de backbone
  - D'autres pensent que les routeurs de Backbone ne peuvent pas effectuer des classifications complexes **des couches 3 et 4**
    - **La classification devrait être effectuée par les routeurs de bordure (Edge Routers)**
    - **Les routeurs de Backbone doivent s'occuper seulement des marquages spéciales**
    - *C'est dans cette direction que les choses se font actuellement le plus*

Fournir une garantie de Débit ? dans un réseau IP (basé sur l'échange des paquets IP) ?

- **Une fois , on a classifié les flux, il faut trouver des mécanismes qui nous permettent obtenir des garanties de débit que l'on souhaite avoir au niveau du réseau.**

- Comment fournir des garanties de débit ?
  - On a un certain nombre de flux qui traversent un routeur pour passer sur la ligne de sortie
  - Si on travaille **avec des liquide**, on peut fournir une garantie de façon très simple :

⇨ Il suffit de regarder le **débit** de chacun des flux d'entrée et de s'assurer que la somme des débits des flux d'entrée soit inférieure à la capacité de la ligne de sortie



$$\sum_{i=1}^{N} I[i] \leq D_{out}$$

Fournir une garantie de bande passante -2-

- Problème : **Dans un réseau, on travaille avec des flux composés de paquets et non pas de liquide (des paquets de taille variable).**
- **Si on a des paquets de taille variable et si on veut supporter des garanties de débit, il va falloir qu'on fasse deux choses :**

1. **Faire attention pour que la liaison de sortie ne soit pas un goulot d'étranglement (⇔ s'assurer que la somme des débits des flux d'entrée soit inférieure à la capacité de la ligne de sortie).**

$$\sum_{i=1}^{N} I[i] \leq D_{out}$$

   - Ça veut dire « en corollaire » qu'on ait un mécanisme qui permet de **limiter le débit du flux entrant**

2. **S'assurer que les buffers des routeurs ne débordent pas (⇔ avoir assez de capacité dans les buffers pour accepter le trafic qui rentre)**
   - Donc, il faut qu'on puisse borner le flux qui rentre aussi bien en terme de débit et en terme d'utilisation du buffer

Limitation du taux des flux entrants

- **Comment limiter le débit d'un flux de paquet IP de tailles variables sur le lien d'entrée ?**
- **Définir un contrat associé au flux qui rentre :**
  1. **Il doit permettre d'exprimer le débit avec lequel, le flux doit arriver en terme de :**
     - ✓ **Nombre d'octets (bits) par unité de temps ?**
  2. **Il doit permettre d'exprimer aussi la quantité de buffer qu'on doit utiliser à l'intérieur du routeur par ce flux.**
- **Sur l'arrivée d'un paquet**
  - Une fois, on a définit ce contrat de trafic, on va pouvoir implémenter un mécanisme qui, pour chaque paquet qui arrive, **doit permettre de vérifier si le paquet respecte le contrat ou non ;**
    - ➔ Si le paquet respecte le contrat, on va l'accepter dans le buffer.
    - ➔ Si non,
      1. **Policing : on va le jeter**
      2. **Shaping : on va le stocker dans un buffer**

token bucket –1-

- **COMMENT DEFINIR ET EXPRIMER CE CONTRAT ?**
  - Le mécanisme le plus simple est le **Token Bucket** « seau à jeton » qui provient des réseaux ATM

- **Définition du Contrat de trafic : Deux paramètres**
  - **R : débit moyen exprimé en octets/sec**
  - **B : taille du bucket « proportionnel à l'utilisation du buffer du routeur »**

- **A chaque flux, on va associer une réserve qui contient des crédits « Tokens ».**
  - Lorsque le flux va démarrer, on va remplir la réserve avec **B tokens**
  - et on va s'arranger pour ramener des nouveaux tokens à l'intérieur de la réserve avec un débit de **R octets/seconde.**
  - ***Le Bucket ne pourra jamais contenir plus de crédits que B (capacité maximale)***

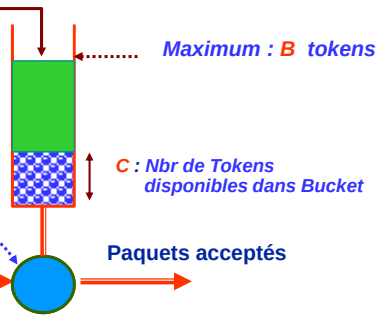
- **A chaque fois un paquet arrive, l'algorithme doit permettre de savoir si ce paquet respecte le contrat de trafic ou non**

token bucket-2-

- **Algorithme :**

```
// Remplissage du Bucket
Au départ C = B
// Toutes les 1/R secondes
   {
      Si (C < B) alors
           C = C +1
   }
```

```
// Arrivée d'un paquet de taille L
   Si (L <= C)
      {
            // Paquet accepté
            C = C – L
      }
            Sinon
      {
            // Paquet non conforme (détruit)
      }
```



R : taux moyen exprimé en octets/sec
B : taille du Token Buckt
// Paquet accepté :
*Pendant T secondes un Bucket accepte au maximum B + (T * R) octets*

token bucket –3-

- **Quand un paquet arrive, Comment faire pour décider si on peut l'accepter ou non ?**

✓ L'algorithme compare la taille de paquet avec le nombre de token qu'il reste dans le bucket

1. Si la **taille de paquet qui arrive < au nombre de Tokens dans le bucket**, alors
   - **le paquet est accepté car se trouve dans le contrat de trafic**
2. Si **pas assez de tokens (crédits) à l'intérieur du bucket pour couvrir la taille du paquet qui arrive,**
   - **le paquet est en dehors de contrat de trafic, on jette le paquet**

- **Propriétés du Token Bucket**

1. Durant une période de **T secondes**, le token bucket va accepter au maximum **B + T*R** octets de trafic
   - **Explication :** Imaginons que le bucket est plein : il contient B octets
   - A l'instant t = 0, on a un paquet de B octets qui arrive, ce paquet est accepté, il va **consommer les B octets** qui se trouvent à l'intérieur du bucket. Ensuite, une fois que bucket sera vide il sera alimenté avec un débit de **R octets /seconde** et donc il ne laissera passer sur la liaison de sortie que **R octets / seconde**
2. Donc, grâce à ça, **on a une borne maximale** sur la **quantité de trafic** qui va être accepté durant une période de temps donnée. Et grâce à cette borne maximale, on va pouvoir dimensionner les buffers de routeur

token bucket-4-

- ***Avantages :***

1. **Permet à la fois :**
   - ✓ **De respecter une valeur de référence (R) et de tolérer une certaine quantité de trafic en excès (B)**
   - ✓ **Peut être utilisé par les fournisseurs pour faire respecter un contrat de trafic, à partir de moment où il fournit une définition algorithmique précise du contrat de trafic, donc on peut savoir pour un paquet qui arrive si:**
     - *Ce paquet est conforme*
     - *Ou bien non conforme*
2. **Fournit une borne maximale sur le taux moyen**
3. **Fournit une borne sur la quantité maximum de trafic accepté pendant une période donnée**
   - ***Permet de fixer la taille des mémoires tampons des routeurs***

token bucket - 5-

- **Inconvénients (en cas de rafales)**
  - **Peut conduire certains flux à consommer plus de bande passante que les autres**
    - *l'accumulation de paquets d'autres flux, paquets transmis ensuite en rafales*
  - Peut augmenter à chaque traversée de routeurs la caractéristique en rafales de certains flux
    - *Correspond à un allongement non désiré des rafales*
    - *Nécessite alors un lissage de trafics en sortie*

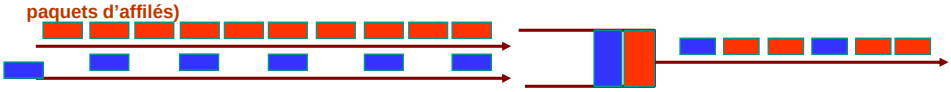
Problème : Effet du multiplexage

- **Dans un réseau, on n'a pas un seule routeur mais un certain nombre de routeurs.**
- **On n'aura pas une seule source, mais plusieurs sources.**
- **Donc, Situation courante :**
  - **On va avoir un certains sources vont être multiplexées sur la liaison de sortie. Ce qui posera des problèmes au niveau de dimensionnement des buffers du routeurs**

Problème : Effet du multiplexage

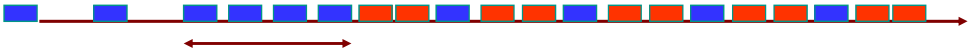
- **Exemple ?**

Flux 1 : faible débit avec R[1]=1/100, B[1] = 10 (un Bucket de taille 10) / mode rafale (il envoie ses paquets d'affilés)



Flux 2 : Haut débit avec R[2]=1/2(moitié de débit de la liaison de sortie, B[2] = 2 (un Bucket de taille petite) il envoie ses paquets régulièrement

- Contrat du trafic de flux en bleu sur le lien de sortie ?



Une rafale importante du flux 2 (ne sont pas conforme au contrat R[2]=1/2) , B[2] = 2

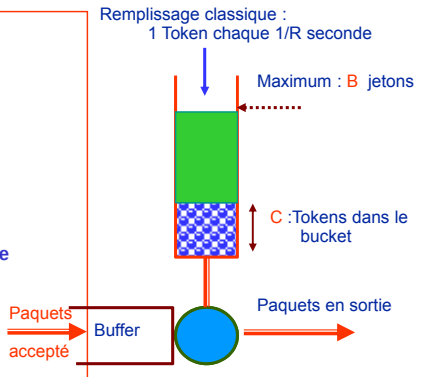
- Constatation : Le fait de multiplexer un flux Haut débit avec un flux bas débit « qui peut avoir des rafales » a engendré des rafale dans le flux haut débit sur la liaison de sortie

- En pratique, plus on va traverser des routeurs intérimaires, plus le caractère en rafale des flux va augmenter => En augmentant les rafales, on va augmenter inévitablement les besoins en buffer à l'intérieur des routeurs

Lissage de trafic « shaping »

- Problème : Pour éviter qu'une source soit perturbé par le multiplexage à travers un certain nombre de routeurs, ce qu'on va faire est de
  - Définir un mécanisme qui permet de mieux réguler le comportement d'une sources (⇔ à la source)
    - Utiliser un Token Bucket modifié (shaper)

```
// Traitement d'un paquet de taille L
Si (L <= C)
{
  // Paquet arrivé dans les temps
  C = C – L;
  Transmet_paquet( );
  }
Sinon
{
  // Paquet arrivé trop en avance
  //Faire attendre le paquet jusqu'au ce qu'il devienne conorme
  Tant que (L > C) {// wait};
  Maintenant, C=L et le paquet est conforme
  C = C – L;
  Transmet_paquet( );
}
```



Le flux en sortie est conforme au contrat défini par R et B
(rafale de taille maximale B)

Lissage de trafic « shaping »-bis-

- **Solution au problème de rafale :** définir un mécanisme qui permet de mieux réguler le comportement d'une source.
  - Pour ça, on va modifier le token bucket de façon pour qu'il puisse être associé à un buffer pour qu'il puisse **réguler le trafic qui arrive en entrée** plutôt que **jeter des paquets** qui en sont pas conformes
  - Grâce au token bucket, on va faire une remise en conformité du trafic / au contrat
- **Contrat de trafic : deux paramètres**
  - R : taux moyen « débit moyen » exprimé en **octets/sec**
  - B : taille du Bucket « proportionnel à l'utilisation du buffer du routeur »
  - On va accoupler au token bucket, un buffer qui servira à stocker les paquets qui arrivent
  - **Lorsqu'un paquet arrive, on compare la taille du paquet avec le nombre B**
    - Si il y a assez de bucket, on transmet directement le paquet sur la liaison de sortie
    - Si non (dans la situation précédente, on jetait le paquet), **on laisse le paquet à l'intérieur du buffer jusqu'à ce que il y a assez de crédit à l'intérieur du token bucket, Ce qui permettra de transmettre le paquet par la suite sur la liaison de sortie.**
      - Avec ça on aura la possibilité de remettre le contrat de trafic en conformité.
      - Évidemment, la remise en conformité dépendra de la taille de buffer qui est à notre disposition.
- Ce mécanisme pourra être utilisé pour **réguler l'accès à un réseau par exemple**.
- **Grâce à ça, on va avoir des mécanismes qui nous permettent de supporter des garanties de débit maximum.**

Contrôle de trafic et QoS dans les réseaux IP

Comment fournir une garantie de débit minimum ?

■ Position du Problème (fournir une garantie minimum ⇔):

- Dans chaque flux, nous avons *maintenant* deux types de paquets :

1. **Des Paquets IP qui font partie de débit minimum garantie(de bande passante) pour le flux *« couverts par la garantie »***
   - ➢ Ces paquets ne peuvent pas être jetés à l'intérieur du routeur
2. **Des Paquets IP qui sont en dehors du minimum garantie de débit (qui seront transmis en Best-Effort)**
   - ➢ Ces paquets devraient être traités comme des paquets **Best-Effort** et peuvent être jetés au besoin pour préserver les garanties

■ Objectifs «Principe» :

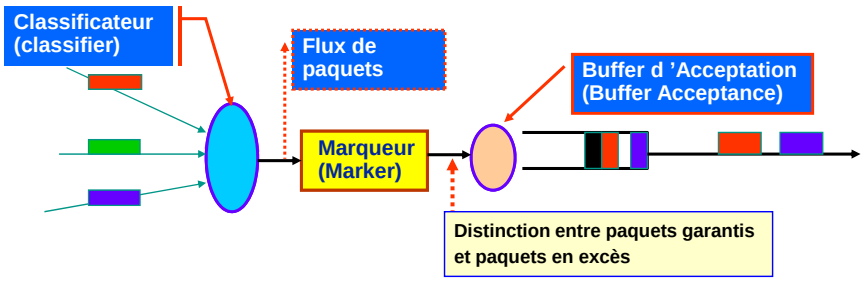
- **Identifier les deux types de paquets**
- **Jeter de préférence les paquets non garanties quand la congestion se produit à l'intérieur du routeur**

Identification des paquets garantis

■ **Principe de la solution :** On va ajouter en **avale du mécanisme de classification**, **un mécanisme de marquage** :

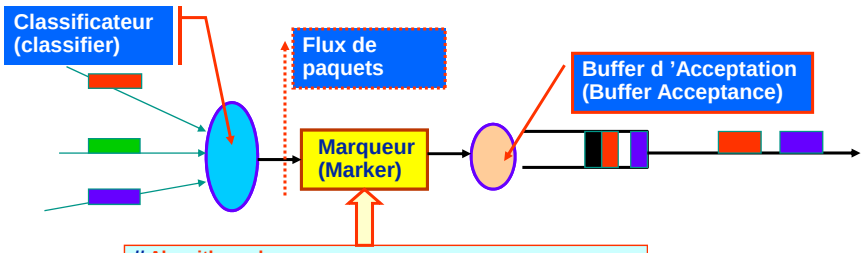
- ➢ Pour identifier les paquets **qui sont garanties** et **les paquets qui ne sont pas garanties**

1. Donc, à la sortie des mécanismes de classification, on va associer à chaque **paquet un identifiant** qui va nous permettre de dire à quel flux le paquet appartient.
2. Connaissant son flux, le paquet va **subir un marquage** dont l'objectif est d'identifier si le paquet fait partie de `débit garantie` ou bien s'il doit être traité en `Best-Effort`



Identification des paquets garantis – marquage probabiliste

- **Marquage probabiliste des paquets – principe**
  - **Mesurer le débit moyen des flux entrants**
  - **Marquer les paquets si le débit moyen dépasse le débit minimal garanti**
  - **Marquer les paquets en excès selon une probabilité P**



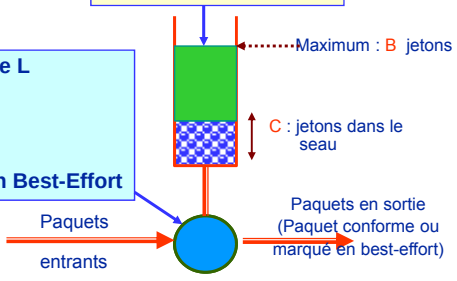

```
// Algorithme de marquage
   Si (débit_moyen <= débit_minimal)
      // Paquet conforme
   Sinon
      P = (débit_moyen - débit-minimal) / débit_moyen
      Tirer un nombre aléatoire X entre 0 et 1
   Si (X < P)
      // Paquet en excès
   Sinon
      // Paquet conforme
```


Marquage Déterministe

- **Principe de base :** La solution la plus simple est de faire ça sur la base d'un **Tocken Bucket**
  - Lorsqu'un paquet arrive, s'il y a assez de crédit on considère que le paquet est garantie
  - Par contre, si pas assez de crédit pour accepter le paquet, il sera considéré en dehors du contrat de garantie et sera marqué comme un paquet Best Effort.
  - Et en fonction de l'importance de paquet on pourra décider **si on veut jeter le paquet ou non en cas de congestion**
  - Nécessite de fixer la hauteur maximale du seau
    - *En plus du débit minimal garanti*
- **Token Bucket modifié**

```
// Remplissage du Bucket
     Au départ C = B
// Toutes les 1/R secondes
     Si (C < B) C = C +1
```

```
// Arrivée d'un paquet de taille L
     Si (L <= C)
          // Paquet conforme
     C = C - L
     Sinon
          // Paquet marqué en Best-Effort
```

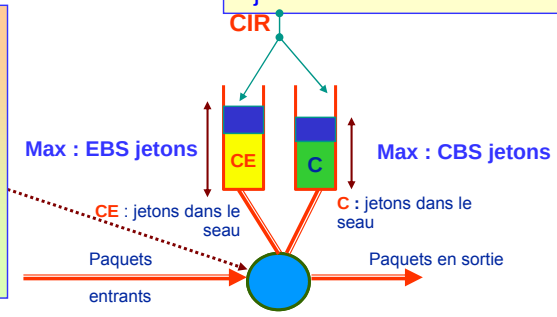


Extension du Tocken Bucket

- Marqueur « à trois couleurs » simple taux
- Un débit et deux tailles de rafales
  - (et deux token bucket en parallèles)
  - CIR : Commited Infomation Rate (débit)
  - CBS : Commited Burst Rate
  - EBS : Excess Burst Size

```
Initialisation
   C  = CBS;
   CE =EBS;
   every 1/CIR second do {
      if(C<CBS)
         { C=C+1; }
      else if (CE<EBS)
         { CE=CE+1 }
      else
         { /* nothing */ }
   }
```

```
Arrivée d'un paquet P de Longueur L
      if (L <= C)
      { /* packet is green */
            C=C-L;
      }
      else if (L <= CE)
      {/* packet is yellow */
            CE=CE-L;
      }
      Else
      {
            /* packet is red */
      }
```



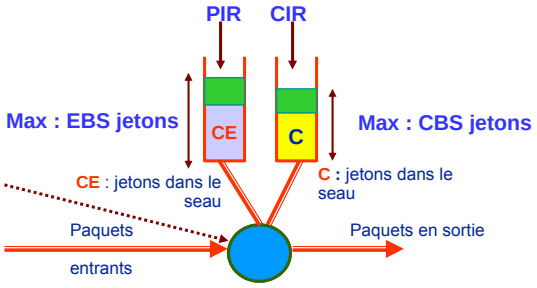
Extension du Tocken Bucket -2-

- Marqueur « à trois couleurs » à deux débits différents (CIR, PIR)
  - CIR : Commited Information Rate
  - PIR : Peak Information Rate
  - CBS : Commited Burst Rate
  - EBS : Excess Burst Size

```
Peak Bucket Filling
Initialisation
   CP=PBS;
   every 1/PIR second do
      {
         if(CP<PBS)
         CP=CP+1;
      }
```

```
Committed Bucket filling
Initialisation
C=CBS;
every 1/CIR second do
   {
      if(C<CBS)
      C=C+1;
   }
```

```
Arrivée d'un paquet P de Longueur L
      if (L < CP)
      {
            packet is red
      }
      else if ( L < C )
      {
            /* packet is yellow */
            CP=CP-L;
      }
      else
      {
            /* packet is green */
            CP=CP-L;
            C=C-L;
      }
```



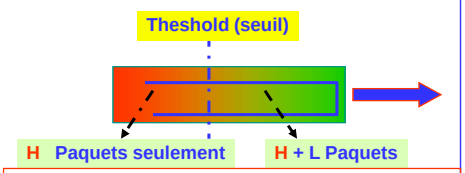
Préférence à la perte des paquets « Paquet Discard Preference »

- **Question : Qu'est qu'on peut faire avec les paquets qui ont été marqués.**
  - Deux types de paquets qui vont avoir des préférences à la perte différents
    1. **Il y a les paquets garantis (qu'on ne peut pas jeter)**
    2. **Et les paquets marqués qu'on peut jeter en cas de problème**

- **Objectif :**
  - **Trouver des mécanismes au niveau du routeur pour jeter préférentiellement les paquets non garantis par rapport aux paquets garantis**

Partial Buffer Sharing

- **Partial Buffer Sharing : implémenté dans tous les switch ATM mais rarement dans les réseaux IP**
- L'idée de base:
  - On va avoir un buffer configuré avec **un seuil**
  - Tant que l'occupation du buffer est inférieur au seuil, on accepte tous les paquets (hautes et basses priorités).
  - Dès que le seuil est atteint : (⇔ on est proche de la congestion) il faut qu'on préserve le peu de ressource à notre disposition pour les paquets garanties



- Dès qu'on a atteint le seuil, on ne jette que les paquets de basse priorité
- **Comment l'implémenter au niveau d'un routeur ?**

```
Arrivée d'un paquet de priorité Haute (H) ou Basse (B)
if (Packet.Type == H)
   { /* paquet est de haute priorité }
        if (Buf.Length < Buf.Size)
                accept_packet( );
        else
                discard_packet( );
   }
   else
   {/*Paquet de priorité Basse */
        if (Buf.Length < Buf.Threshold)
                accept_packet( );
        else
                discard_packet( );
   }
```

Destruction des paquets à la priorité - 2-

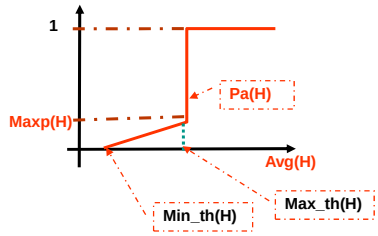
- Implémentation la plus populaire : **RED Pondéré** « Weighted RED »
  - **Est une extension de RED pour supporter N niveaux de priorité de perte des paquets (N Packet Discard Prefernces)**
- Principe de la solution : N algorithmes RED exécutés en parallèle
  - Le premier décide l'acceptation des paquets de priorité N, qui devrait être jetés seulement en cas de congestion sévère.
  - Le second décide l'acceptation des paquets de priorité N-1 qui devraient être jetés plus tôt que les paquets de priorité N
  - ….
  - Le nième algorithme RED décide de l'acceptation des paquets sans aucune priorité

Variantes de RED

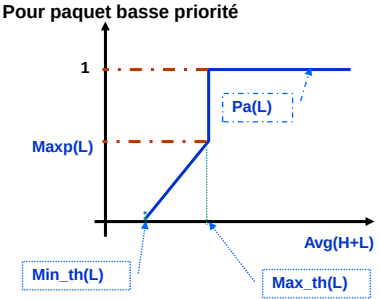
RED Pondéré « Weighted RED » -1-

- RED avec Entrée / Sortie (RED In/Out » (⇔ RIO)
  - Proposition initiale pour supporter deux priorités à la perte
  - Principes
    1. Calcul de deux valeurs moyennes de l'occupation du buffer: **Avg(H)** et **Avg(H+L)**
    2. Appliquer un algorithme **RED** conservatif pour des paquets de **Haute Priorité** avec des **seuils grands**
    3. Appliquer un algorithme **RED** agressif pour les paquets de **faible priorité** avec de **petits seuils**

Probabilité à la perte
Pour paquet haute priorité



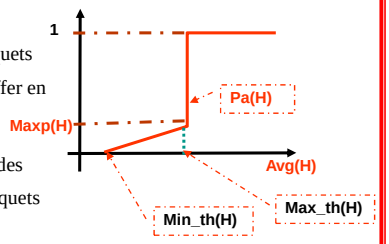
Probabilité à la perte
Pour paquet basse priorité



RED Pondéré « Weighted RED »-2-

- **Idée ; on demande à RED de calculer deux valeurs moyennes de l'occupation de buffer :**
  - Une valeur moyenne qui va prendre en compte les paquets de haute priorité **Avg(H)**
  - Une deuxième valeur moyenne qui va prendre en compte tous les paquets **Avg(H+L)**
- **Ensuite, deux algo RED :**
  1. Un pour les paquets de **haute priorité (algo très conservatif)** ,
     - il va rarement jeter des paquets
  2. Un autre pour les paquets de **basse priorité (algo agressif)** pour les paquets non garantie ;
     - beaucoup plus agressif au niveau de jeté des paquets

- **algo très conservatif :**
  - On fixe des **seuils min et max élevés** de façon à ne pas jeter des paquets rapidement et ce seuil s'applique uniquement sur l'occupation du buffer en taille moyenne des paquets **de haute priorité**.
  - Ensuite, on va **fixer une valeur maximale de probabilité** à la perte des paquets de haute priorité relativement faible pour ne pas jetter des paquets de Haute priorité rapidement

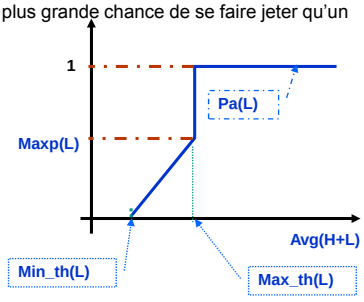


RED Pondéré « Weighted RED »-3-

- **Pour les paquets de basses priorités :**
  - On va décider de jeter des paquets sur base d'occupation moyenne du buffer complet en **prenant en compte les paquets de haute priorité et les paquet de basse priorité**

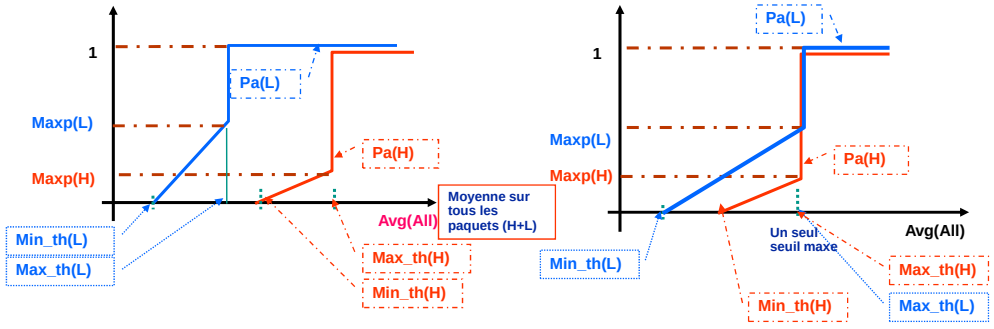
⇒**si le buffer est fort occupé des paquets de haute priorité**, les paquets de basse priorité n'auront aucune chance de rentrer dans le buffer.

  - On fixe des **seuils min et max** pour ces paquets qui vont être plus faibles que ceux utilisés pour les paquets haute priorité
  - et on aura une probabilité de perte maximale qui va être plus importante pour les paquets de Basse priorité.
  - De cette façon lorsqu'un paquet de basse priorité arrive, il aura une plus grande chance de se faire jeter qu'un paquet haute priorité



RED Pondéré « Weighted RED » avec N priorités à la perte

- **RED Pondéré « implémenté sur les CISCO »**
  - Peut être employé pour supporter **N niveaux de des priorités à la perte**
  - Principes :
    1. Calculez **une seule occupation moyenne pour tous les paquets** dans le buffer
    2. Algorithme RED conservatif pour les paquets de priorité élevée
       - valeurs importantes pour min_th(H) et max_th(H)
    3. Algorithme ROUGE agressif pour les paquets de faible priorité
       - petites valeurs pour min_th(L) et min_th(L), et prob de perte plus élevé .



Contrôle de trafic et QoS dans les réseaux IP

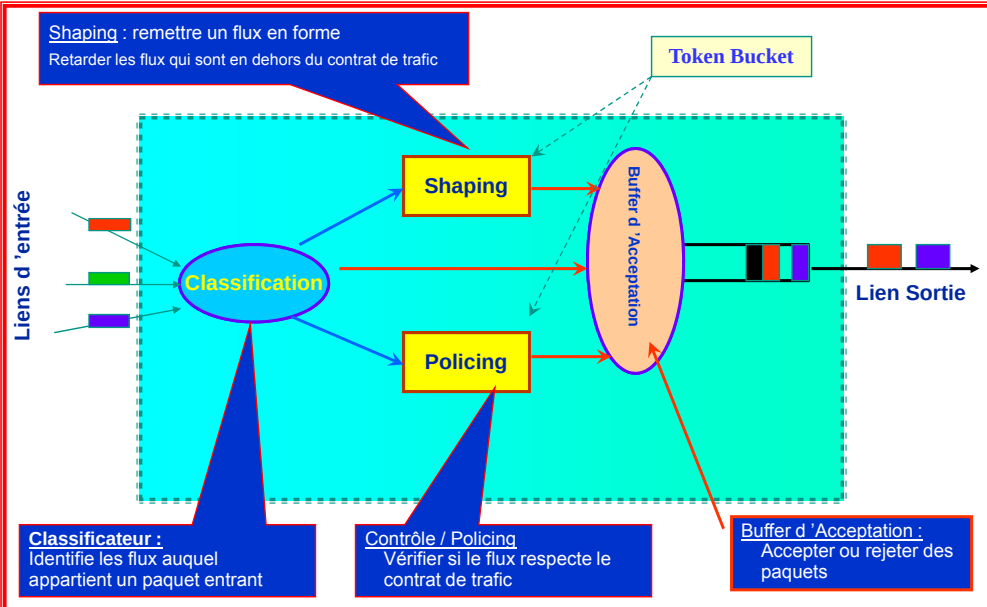
1. Applications
2. Mécanismes de contrôle de trafic de Paquet-niveau
   - ⇨ Service Best-effort
   - ⇨ Service dé débit maximum garantie (⇔ débit)
   - ⇨ Service dé débit minimum garantie (⇔ débit)
   - ⇨ Garanties de délai
3. Services Normalisés
4. Contrôle de trafic d'Intra-domaine
5. Contrôle de trafic d'Inter-domaine
6. Études de cas

Vers des réseaux multiservices

♦ Problèmes :

- **Comment faire cœxister sur une seule liaison de sortie à travers un routeur un trafic best effort avec un trafic appartenant à des flux respectant une certaine garantie**
  - ⇔ **Les paquets « garantis » ne devraient pas être perturbés par les paquets Best-effort**
  - ⇔ **Les paquets Best-effort devraient pouvoir utiliser les liens de sortie en absence du trafic garanti**
- **Comment pouvoir fournir différentes garanties au délais**
- **Est il possible de pouvoir faire ceci avec le routeur simple que nous avons proposé ou bien il faut lui ajouter d'autres mécanismes**

Routeur simple v3

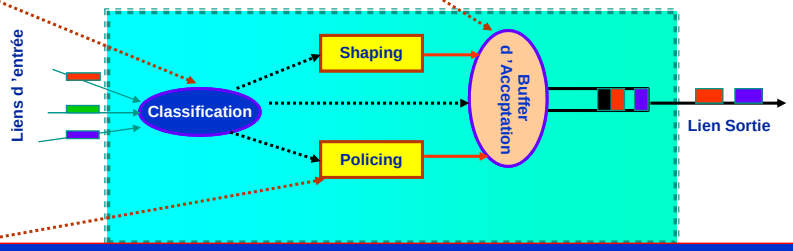


Multiplexage avec le routeur simple - v3 -

- **La question est de savoir si sur base de ces mécanismes si on peut supporter efficacement un réseau *multi-services***

Identifie les trois types de flux
Ajouter un identificateur interne
dans les paquets des flux

Suppression des paquets de faibles priorités
avant les paquets de priorité plus grande



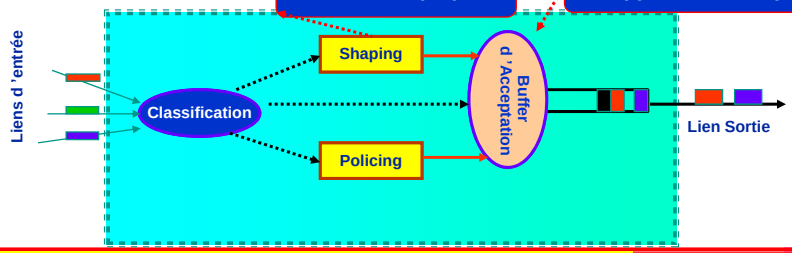
❑ **Marquage basé sur le type de flux (flux_id) et le débit actuel**
- Les paquets Best-effort sont marqués avec une basse priorité
- Pour un flux de débit minimum garanti :
  - Les paquets en excès sont considérés avec une priorité faible
  - Les paquets respectant le contrat de débit minimum garanti ont une priorité plus grande
- Les paquets de débit maximum garanti sont marqués avec une priorité plus grande

Au niveau des Garantie des délais ?

- **Mécanismes supportés par le routeur simple -V3-**
  1. **Au niveau du shaping**, on a un token bucket qui nous permet de retarder un paquet si le paquet arrive trop vite par rapport au contrat de trafic, **mais ça ne permet pas d'avancer un paquet par rapport à un autre.**
  2. **Le mécanisme d'acceptation de paquet**, nous permettrait de jeter un paquet.
     - **Ça ne nous permettrait pas d'avancer un paquet par rapport à un autre**
- **Avec ce routeur,** on n'a pas assez de mécanismes à notre disposition pour faire une différentiation au niveau de délai entre les paquets de différents flux.
  - On peut faire une **différentiation négative** en retardant certains flux mais on peut pas **faire différentiation positive**

Retarder les paquets

Suppression des paquets

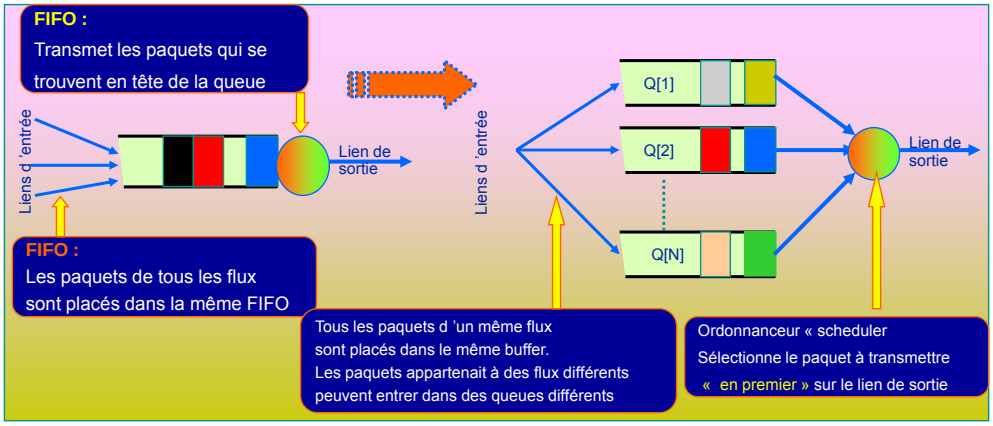


Garantir les délais de transmission -2-

- Problème posé :
  - Comment différencier ?
    - → Les paquets nécessitant des délais de transmission faibles
      - ❖ **Exemple : voix sur IP, application en flux continu, etc.**
    - → Des paquets sans contraintes de délai de transmission
      - ❖ **Exemple : transfert de fichier, requête HTTP**
- **Solution générale :**
  - Remplacer l' unique buffer FIFO des interfaces de sortie par ***un ensemble de files d'attente***
    - Tous les paquets d'un flux donné seront placés dans la même file d'attente de sortie
    - Les files sont gérés par un « Ordonnanceur » (scheduler) permettant de différencier les files entre elles
    - Sélectionner les flux qui doivent avoir des délais plus faibles **prioritairement** par rapport aux autres flux

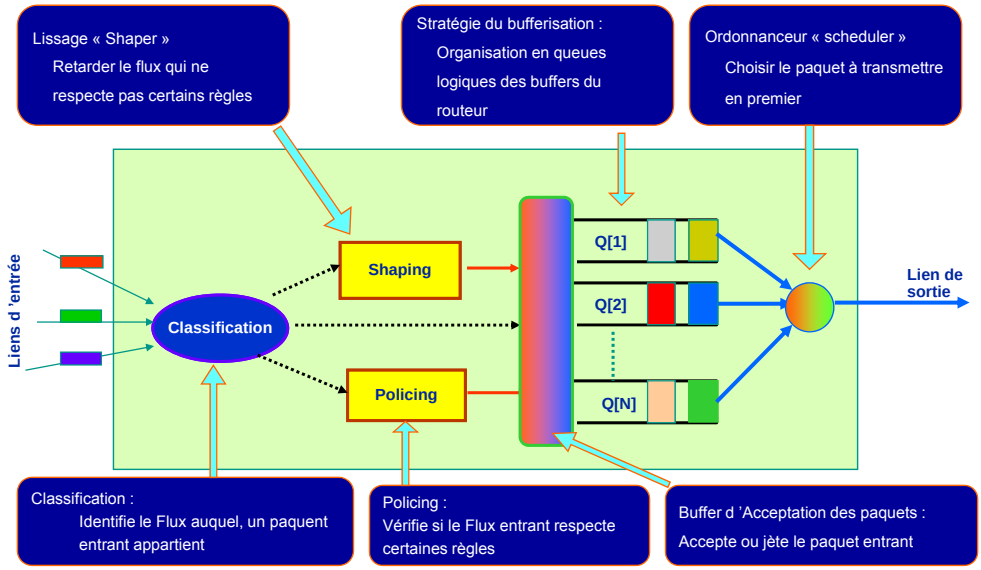
Garantir les délais de transmission –3-

- **Pour faire une différentiation positive :**
  - ✓ il faut qu'on ajoute **un scheduler** avec un **ensemble de queues** comme on a fait pour le service Best-Effort,
  - ✓ **mais on déterminera autrement comment il faut placer les paquets dans les queues**.



Routeur possédant des capacités de QoS

Notre routeur devient quasiment complet. Il contient les mécanismes ci-dessous



Routeur possédant des capacités de QoS-commentaires

- **Fonctions remplies par les différentes parties**

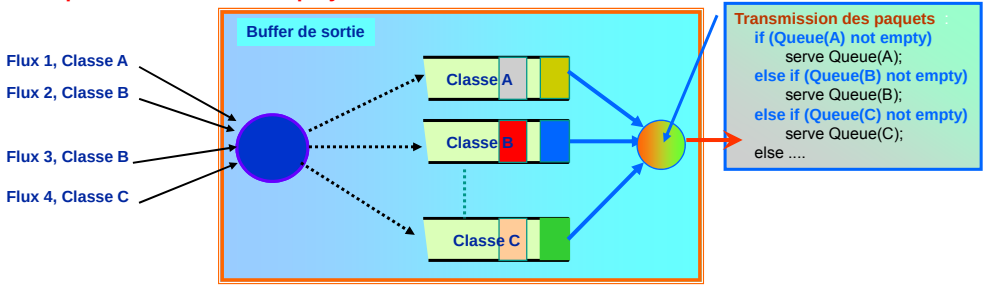
1. **La classification est important pour pouvoir identifier les paquets :**
   - Flux best-effort / flux avec une garantie de débit minimal / flux avec garantie maximale
   - La classification va ajouter **un identifiant interne** au paquet pour savoir à l'intérieur du routeur à quel flux le paquet appartienne
   - **En fonction du résultat de la classification**, on enverra le paquet au mécanisme de Shaping, au mécanisme de policing ou bien directement au mécanisme d'acceptation des paquets dans le buffer
2. **Le Shaping :** va nous permettre de retarder les flux qui ne respectent pas leur contrats de trafic
3. **Le policing : permet de marquer (ou bien de jeter) les paquets qui sont en dehors du contrat de trafic**
   - **Flux Best-Effort** : paquets marqués en basse priorité
   - **Flux avec garantie minimale** : marquage des paquets avec une priorité haute ou une priorité basse suivant que le paquet qui arrive se trouve à l'intérieur du contrat de trafic ou non
   - **Flux avec débit maximum** : les marquer avec une haute priorité
4. **Ensuite, tous les paquets, vont passer dans le mécanisme d'acceptation des paquets dans le buffer :**
   - **Il permet de décider si oui ou non le paquet peut rentrer dans le buffer**
5. **Ensuite, on va avoir des buffers qui vont être organisés sous la forme d'un ensemble de queues logiques**
   - **Différentes stratégies d'organiser les queues à l'intérieur du routeur**
6. **Le scheduler**
   - qui nous permet de sélectionner le paquet à **transmettre en premier sur la liaison de sortie**
   - **En jouant sur le scheduller, on pourra introduire une différentiation positive au niveau de délai entre les différentes queues du routeur**

Ordonnanceur « scheduler »

- Objectif de l'Ordonnanceur :
  - Parmi toutes les queues logiques contenant au moins un paquet, sélectionner le paquet qui devrait être transmis sur le lien de sortie
- L'ordonnanceur doit :
  1. Etre facile d'implémentation dans le matériel
  2. Supporter le service Best-Effort et les Services Garantis
  3. Fournir l'équité au trafic Best-Effort
     - Equité Max-Min est le but désiré
  4. Fournir la protection et isolation entre les flux
     - ⇔ Un flux ne devrait pas monopoliser toutes les ressources et la bande passante au détriment des autres flux
  5. Fournir des garanties :
     - ⇔ En terme de débit, des délais. Des garanties probabilités ou déterministes en fonction du type de scheduler

Ordonnanceur à priorité « Priority based Scheduler »

- **Avantages :**
  - Facile à mettre en application
  - Les paquets dans la classe prioritaire élevée voient des délais faibles
- **Inconvénients :**
  - Il n'offre **aucune protection** (un utilisateur qui fait de trafic ou autre dans la queue de haute priorité, ce flux va rester dans cette queue et il n y a plus rien qui passera)
    - *Il faut bien sélectionner ce que vous allez mettre dans la queue de Haute priorité)*
  - Un flux de priorité élevée peut absorber toutes les ressources
- **La priorité devrait être employée avec soin...**

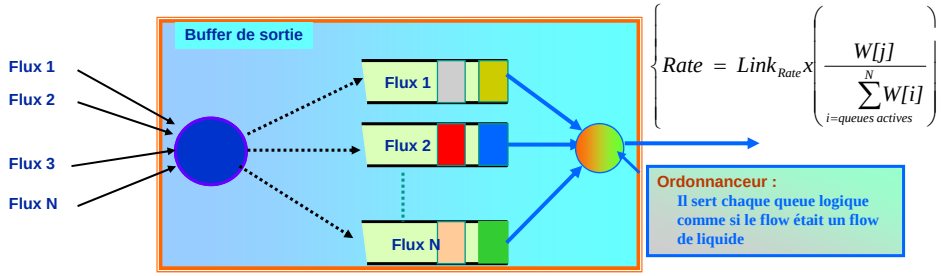


- **Il ne répond pas à nos besoins de support efficace du trafic Best-Effort**

GPS : Generalized Processor Sharing –1-

♦ **Ordonnanceur Idéal :**

- A chaque queue logique queue[i], on associe un poids **W[i]**
- Chaque queue logique est visitée par l'Ordonnanceur et servi comme s'il contenait un **flux liquide**
- Au temps **t**, la **Queue[j]** est servi au taux suivant :



♦ Donc, à tout instant t, la queue n° j sera servi avec un débit = **débit de la ligne de sortie * le rapport entre (le poids associé à la queue j et la somme des poids des queues qui sont actifs à cet instant)**

⇔En fonction du poids associé à chacune des queues, **on pourra affecter une importance à chacune en terme d'utilisation de la liaison de sortie.**

- **La répartition sur la liaison de sortie sera au prorata du poids associé à chaque queue**.

⇔En fonction des poids, on pourra associer une **différenciation en terme de débit, ou bien en terme de délai**.

GPS : Generalized Processor Sharing - 2-

■ **Caractéristiques de GPS :**

1. **Fournit aux flux des garanties en terme de débit (il suffira de configurer correctement les poids de scheduler). Cette garantie est valable au travers**
   - *d'un seul scheduler GPS Ou bien , à travers un réseau de N shceduler GPS*
2. **On peut montrer qu'on peut fournir des garanties en terme de délai pour un flux à condition de travailler avec des flux qui sont limité par un Token Bucket (⇔ mis en conformité par un token bucket (R,B))**
   - A travers un seul GPS
   - Ou bien , à travers un réseau de N scheduler GPS

   ⇒ $\left\{ Délai\ \max = \frac{B}{R} \right.$

   => le délai qu'on va obtenir pour un flux va dépendre uniquement des paramètres du flux (ne dépendra pas du nombre de schedulers traversés, ni du comportement des autres flux du réseau)
3. **Fournit une limite sur l'utilisation du buffer** ⇒ $\left\{ Capacit\ \max\ du\ buffer = B \right.$
4. **Assure une protection entre les différents flux**
   - *Un flux ne peut pas compromettre les garanties allouées pour un autre flux*
5. **Au niveau de variation de délai, la seule garantie est d'avoir une « variation de délai » comprise entre ([0,D_{max} = B/R])**

■ **Inconvénient : Un Ordonnanceur idéal ⇔ n'est pas implémentable**

WRR : Weighted Round Robin « A tour de rôle Pondéré »

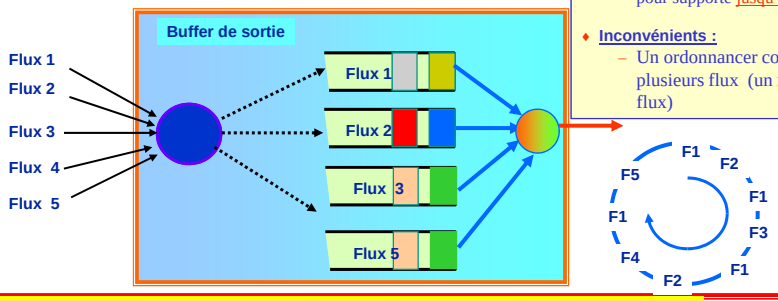
❑ **Problème posé**
- Comment différencier les différents flux en **offrant plus de débit à certains** de ces flux ?

❑ **Solution standard :**
- Dans un même « tour », visiter une ou plusieurs fois la même file d' attente
- En fonction du nombre de positions à l'intérieur du cercle que la queue aura, elle aura plus d'opportunité d'être transmise plus rapidement
- **Exemple :**
  - 4/9 de la bande passante à F1,
  - 2/9 à F2
  - 1/9 à F3, F4 et F5

o *Visiter les files dans l'ordre cyclique suivant*
- F1, F2, F1, F3, F1, F2, F1, F4, F1, F5, etc.

♦ **Avantages :**
- Facile à implémenter
- Plusieurs poids offrent la possibilité d'allouer des bandes passantes différents
- Protection entre les flux
- Extension possible de Deficit Round Robin pour supporte jusqu'au 8 flux

♦ **Inconvénients :**
- Un ordonnancer complexe pour supporter plusieurs flux (un nombre important de flux)



WFQ : Weighted Fair Queuing « Attente Equitable Probable »

❑ **Objectif :**
- Fournit une implémentation « approximée » de GPS

❑ **Idée :**
- Simuler GPS sur la base de paquet par paquet
- Servir les paquets « approximativement » dans l'ordre dans lequel, ils auraient dû être servi avec un GPS

❑ **Comment faire ceci ?**
1. Lorsqu'un paquet arrive, Calculer le temps avec lequel GPS aurait dû servir chaque paquet (temps de fin)
2. Mettre ce temps à l'intérieur de paquet
3. Servir les paquets dans l'ordre croissant de de leurs temps de fin avec le scheduler GPS
4. **Prend en compte la taille des paquets**

❑ **Chez Cisco : WDRR qui est implémenté (pend en compte la taille des paquets aussi)**

Ordonnanceur Virtual Clock

♦ Principe de base :

1. Associer une **estampille temporelle(TS)** à chaque paquet entrant (même idée que WFQ)
   – *En fonction de la bande passante allouée à sa file d'attente*
2. Sélectionner à chaque cycle
   – *Le paquet avec l' estampille la plus faible*

♦ Premier algorithme

- Pour chaque queue i, réserver un débit **D[i]** :
- **V[i] :** variable d'état de la file « i » (⇔ associé à chaque queue)
- A chaque arrivée dans la file « i » d'un paquet de taille P

  V[i] = V[i] + (P / D[i]) ; ⇔ le temps auquel le paquet aurait été servi (ou aurait fini d'être servi) si on avait une liaison **TDM (Time Division Multilexing)**

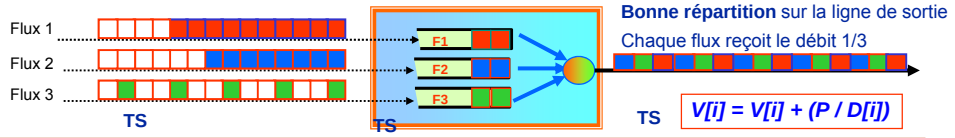
  Estampiller le paquet avec V[i]

♦ Ordonnanceur :

- Sélectionner et transmettre le paquet qui a la **plus petite Estampille** « TimeStamp »

Ordonnanceur Virtual Clock - Exemple (1)

- On suppose des paquets de **taille unitaire**
- Le débit réservé à chacun des flux est **le tiers de celui de la liaison de sortie**
- Trois flux qui ont des comportements différents :
  - Flux 1 et Flux 2 transmettent **une rafale**
  - Flux 3 transmet exactement **à son débit réservé** (1/3 de débit de sortie)

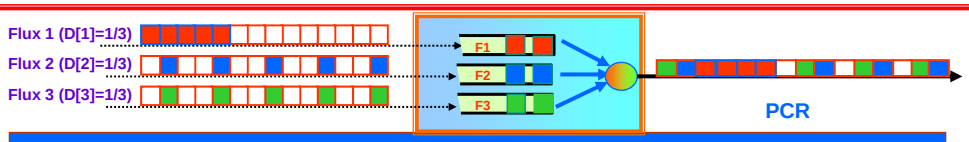


Bonne répartition sur la ligne de sortie
Chaque flux reçoit le débit 1/3

V[i] = V[i] + (P / D[i])

T	V(F1)	Q(F1)	V(F2)	Q(F2)	V(F3)	Q(F3)	Ordonnancé
0	0+3	3	0+3	3	0+3	3	F1
1	3+3	6	3+3	3, 6	3	3	F3
2	6+3	6,9	6+3	3, 6, 9	3	-	F2
3	9+3	6, 9,12	9+3	6, 9, 12	3+3	6	F1
4	12+3	9, 12, 15	12+3	6, 9, 12, 15	6	6	F3
5	15+3	9, 12, 15, 18	15+3	6, 9, 12, 15,18	6	-	F2
6	18+3	9, 12, 15, 18, 21	18+3	9, 12, 15, 18, 21	6+3	9	F1
7	21+3	12, 15, 18,...	21+3	9, 12, 15, 18, 21	9	9	F3
8	24+3	12, 15, 18,...	24+3	9, 12, 15, 18,...	9	-	F2
9	....						

Ordonnanceur Virtual Clock - Exemple (2)

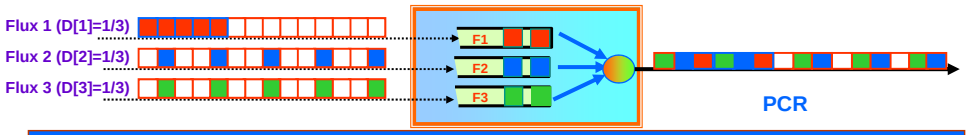


T	V(F1)	Q(F1)	V(F2)	Q(F2)	V(F3)	Q(F3)	Ordonnancé
0	0	-	0+3	3	0+3	3	F2
1	0	-	3	-	3	3	F3
2	0	-	3	-	3	-	-
3	0	-	3+3	6	3+3	6	F2
4	0	-	6	-	6	6	F3
5	0	-	6	-	6	- -	
6	0	-	6+3	9	6+3	9	F2
7	0	-	9	-	9	9	F3
8	0	-	9	-	9	-	-
9	0+3	3	9+3	12	9+3	12	F1
10	3+3	6	12	12	12	12	F1
11	6+3	9	12	12	12	12	F1
12	9+3	12	12+3	12,15	12+3	12,15	F1
13	12+3	15	15	12,15	15	12,15	F2

On constate des flux qui se comportent normalement qui voient leur débit affecté par un flux qui se comporte annormalement.
Les flux 2 et 3 sont à la limite pénalisé d'avoir transmis des paquets au début.
Le problème vient du fait que à chaque fois, on reprend le point de départ (t=0) pour le calcule de la nouvelle valeur de Time Stamp

Ordonnanceur Virtual Clock - Exemple (3)

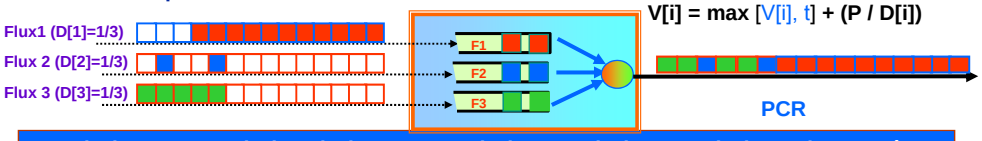
- **Problème posé par l'exemple précédent :** En cas de rafales, il est montré que
  - Certains flux peuvent **accaparer** momentanément les ressources de la ligne de **sortie**
- **Solution proposée : Autres fonctions de mise à jour de V[i]**
  - **V[i] = max** [V[i], t] **+ (P / D[i])** / Avec « t » le temps présent « courant »



T	V(F1)	Q(F1)	V(F2)	Q(F2)	V(F3)	Q(F3)	Ordonnancé
0	0	-	max(0,0)+3	3	max(0,0)+3	3	F2
1	0	-	3	-	3	3	F3
2	0	-	3	-	3	-	-
3	...						
9	max(0,9)+3	12	max(9,9)+3	12	max(9,9)+3	12	F1
10	max(12,10)+3	15	12	12	12	12	F2
11	max(15,11)+3	15,18	12	-	12	12	F3
12	max(18,12)+3	15,...	max(12,12)+3	15	max(12,12)+3	15	F1
13	max(21,13)+3	18,...	15	15	15	15	F2

Ordonnanceur Virtual Clock - Exemple (4)

Un autre exemple :



T	V(F1)	Q(F1)	V(F2)	Q(F2)	V(F3)	Q(F3)	Ordonnancé
0	max(0,0)+3	3	0	-	0	-	F1
1	max(3,1)+3	6	0	-	0	-	F1
2	max(6,2)+3	9	0	-	0	-	F1
...							
8	max(24,8)+3	27	0	-	0	-	F1
9	max(27,9)+3	30	max(0,9)+3	12	max(0,9)+3	12	F2
10	max(30,10)+3	30,33	12	-	max(12,10)+3	12,15	F3
11	33	30,33	12	-	max(15,11)+3	15,18	F3
12	33	30,33	max(12,12)+3	15	max(18,12)+3	18,21	F2
13	33	30,33	15	-	max(21,13)+3	18,21,24	F3
14	33	30,33	15	-	max(24,14)+3	21,24,27	F3
....							

L'idée de faire démarrer le le calcul de TS au temps présent c'était une bonne idée mais pas suffisante.
En effet, on constate que Flux 1 a consommé de débit sur la ligne de sortie alors qu'il était seul à transmettre

SCFQ : Self Clocked Fair Queueing (switch ATM : Virtual Spacing)

APPROXIMATION DE GPS :

Principe :

- Pour éviter la situation précédente : Il faut non seulement prendre en compte la valeur de temps actuel, mais aussi la valeur de time-stamp de autres sources
- L 'ordonnanceur choisit le paquet qui a la plus petite estampille
- Associer une estampille à chaque paquet arrivant

Algorithme :

D[i]: débit associée à Queue[i]

V[i]: variable d'état associés à la Queue[i]

V: variable d'état associé au scheduler [V= TS du paquet qu'on est en train de transmettre]

V représente l'horloge interne du scheduler (qui n'est pas forcément alignée sur l'horloge temps réel, elle pourra aller plus vite)

A l'arrivée d'un paquet de taille P dans Queue[i] ;

V[i] = max(V[i], V) + (P/D[i])

V[i] est associé au paquet arrivé

Ordonnanceur : Choisir le paquet qui la plus petite estampille

Garanties –1-

- **Quels sont les garanties fournies par ces Schedulers ?**
- **On a vu ;**
  - **GPS**
  - **WFQ**
  - **SCFQ**
  - **Deficit-WRR**
- **Ils fournissent des garanties en terme de débit et de protection entre les flux**
  - **Ces garanties sont indépendantes du comportement des autres flux**
    - ⇔ Une bonne protection au niveau des schedulers et un flux ne peut pas s'attribuer de façon malhonnête les ressources qui sont associées normalement à un autre flux
    - ⇔ Ceci nécessite bien sur qu'on ait un ***bon mécanisme d'acceptation de paquets*** dans les buffers

Garanties -2-

- **Ces schedulers fournissent aussi des garanties en terme de délais :**
  - Pour que ces garanties en terme de délais soient possibles, il faut qu'on ait des flux dont le comportement est borné par un **Token Bucket (R, B)**
    - ⇔ Donc, seulement pour les flux mis en conformité par un token bucket (R,B)
    - ⇔ la garantie ne dépend pas du comportement d'autres flux

Garanties -3-

Pour GPS : $\{GPS : délai\ \max = \frac{B}{R}$

- **Car ici, on transmet des paquets et non pas de fluide.**
- **Exemple : un paquet qui devrait être transmis rapidement mais est arrivé juste au moment ou on a commencé à transmettre le premier bit d'un autre paquet, donc il faut attendre que ce gros paquet soit transmis sur la ligne de sortie. Donc, au niveau de délai, on a une borne plus grande par rapport à GPS**
- **L'allure de cette borne ?**

1. Pour WFQ :

Terme qui provient de GPS

$$\{WFQ : délai\ \max = \frac{B}{R} + \frac{N \times P_{\max}}{R} + \sum_{i=1}^{N} \frac{P_{\max}}{C_i}$$

- Terme pour prendre en compte qu'on peut se trouver juste dernier un gros paquet.
- Le cas pire est lorsqu'on veut transmettre juste après un gros paquet dans chaque scheduler.
- Donc on va attendre **Pmax/R** dans chaque scheduler

- Transmission des paquets sur des lignes de débit fini
  - **Temps de transmission sur les lignes**

Pmax = Taille max de paquet

Garanties -4-

2. Pour Virtual Clock : même garantie en terme de débit que WFQ

- Terme pour prendre en compte qu'on peut se trouver juste dernier un gros paquet.
- Le cas pire est lorsqu'on veut transmettre juste après un gros paquet dans chaque scheduler.
- Donc on va attendre **Pmax/R** dans chaque scheduler

Terme qui provient de GPS

$$\{Virtual\ Clock \quad délai\ \max = \frac{B}{R} + \frac{N \times P_{\max}}{R} + \sum_{i=1}^{N} \frac{P_{\max}}{C_i}$$

- Transmission des paquets sur des lignes de débit fini
  - **Temps de transmission sur les lignes**

- **Par contre pour Virtual Clock, on n'a pas de garantie de débit (exemples précédents)**

Garanties -5-

3. Pour SCFQ : garantie de délai assez mauvaise

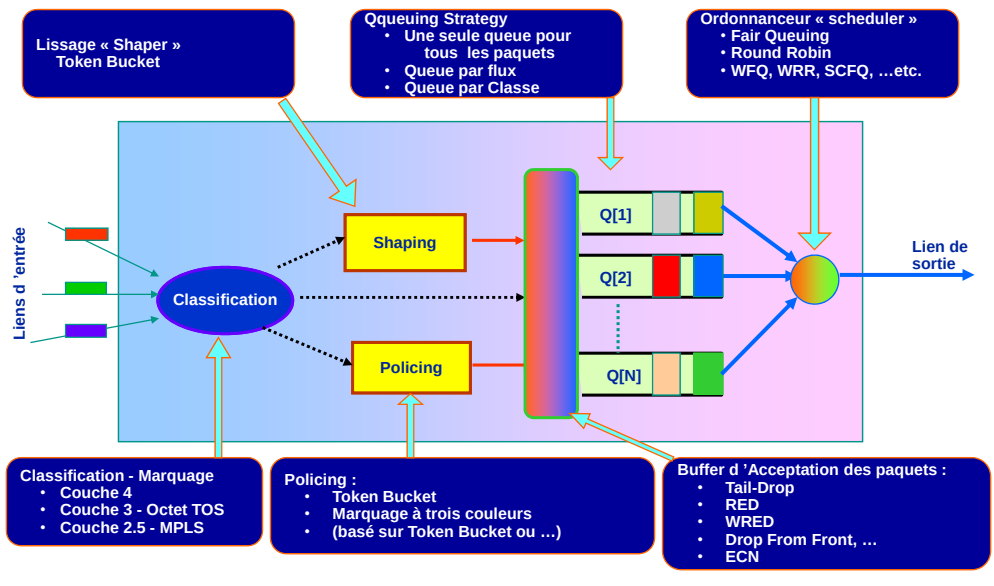
- Terme pour prendre en compte qu'on peut se trouver juste terrier un gros paquet.
- Le cas pire est lorsqu'on veut transmettre juste après un gros paquet dans chaque scheduler.
- Donc on va attendre **Pmax/R** dans chaque scheduler

Terme qui provient de GPS

$$\left\{ SCFQ \ : \ délai \ \max \ = \frac{B}{R} + \frac{N \times P_{\max}}{R} + \sum_{i=1}^{N} \frac{K_i \times P_{\max}}{C_i} \right.$$

- **Transmission des paquets sur des lignes de débit fini**
  - **Temps de transmission sur les lignes**
- **Ki est le nombre de flux qui passe pas le scheduler N° i**

Architecture d'un routeur gérant la QoS au niveau des paquets. « Mécanismes de contrôle de trafic au niveau paquet »



FIN

Références

1. http://www1.cs.columbia.edu/~library/TR-repository/reports/reports-2004/
2. http://www.rennes.enst-bretagne.fr/~toutain/G6Recherche/Papier-Intserv-Toutain.PDF
3. http://etr05.loria.fr/slides/vendredi/ETR2005_Mammeri.pdf
4. http://tel.ccsd.cnrs.fr/docs/00/04/90/98/PDF/tel-00011034.pdf
5. http://www.loria.fr/~akoubaa/PhD/Chapitre1.pdf

Contrôle de trafic et QoS dans les réseaux IP

1. Applications
2. Mécanismes de contrôle de trafic au niveau paquet
   - ⇨ Service Best-effort
   - ⇨ Service de bande passante (⇔ débit) maximum garantie
   - ⇨ Service de bande passante (⇔ débit) minimum garantie
   - ⇨ Garanties de délai
3. Services Normalisés :
   1. Services Intégrés « Integrated Services : IntServ»
      - ❖ Architecture
      - ❖ RSVP : Ressource Reservation Protocol
        - ▪ **Les services Garanties :**
        - ▪ **Service Charge Contrôlée « Controlled Charge »**
        - ▪ **Service Nul « Null Service »**
   2. Services Différenciés « Differnciated Services : DiffServ »

- Contrôle de trafic d'Intra-domain
- Contrôle de trafic d'Inter-domain
- Études de cas