Ordonnancement

Marceau Coupechoux
UE RES222 « Accès au Médium et Ordonnancement »
Télécom ParisTech
Département Informatique et Réseaux

Plan

Introduction

- Les réseaux actuels doivent servir des applications ayant des exigences de **qualité de service** diverses.
- Exemples : Voix, vidéo-conférence, imagerie médicale, transferts de fichiers, télévision, navigation Internet, etc.

- Les réseaux IP/Ethernet ont été conçus essentiellement pour transporter des services « au mieux » (***best effort***).
- La nécessité existe donc de développer des algorithmes capables de garantir des qualités de services variées.

- Un outil utile pour atteindre ce but est la **discipline de service** implémentée au niveau des commutateurs (*switch*) et routeurs.
- Une discipline de service détermine l'ordre de transmission des paquets et contrôle les interactions des connexions entre elles au niveau d'un commutateur.

Gestion de trafic
Qualité de service

- **La gestion de trafic** (*trafic management*) a pour but d'utiliser de manière efficace les ressources disponibles tout en assurant une qualité de service à chaque connexion préalablement négociée.

- **Principaux critères** de qualité de service (QoS) :
  - Débit,
  - Délai de bout en bout,
  - Gigue : différence de délai maximale entre deux paquets (cas idéal=gigue nulle, c.a.d. le réseau n'introduit qu'un délai constant) ou variance du délai. Si le récepteur connaît une borne sur la gigue, il peut calculer la taille de la mémoire nécessaire pour éliminer cette gigue.
  - Pertes de paquets : ceci peut se produire soit à cause d'une congestion, soit à cause d'un délai trop important (exemple : la voix).
  - Disponibilité des équipements.

Gestion de trafic
Qualité de service

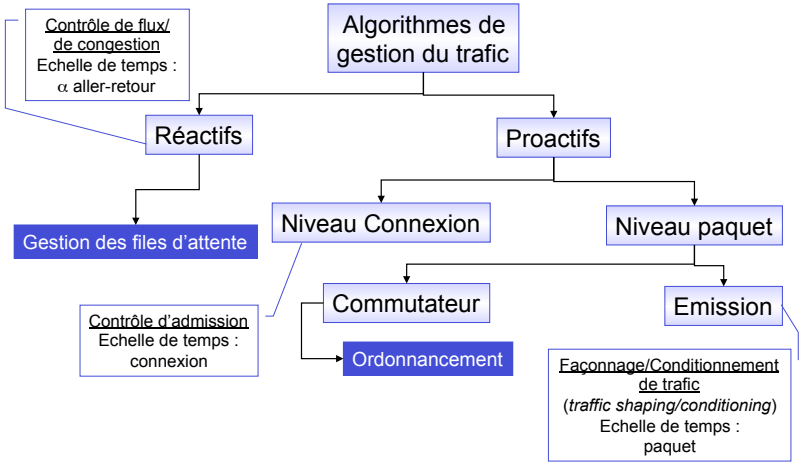
- La définition d'un contrat avec QoS nécessite la définition de :
  - Service Level Agreement (SLA) : définit les cibles en termes de QoS pour les types de trafic concernés,
  - Trafic Conditioning Agreement (TCA) : ce sont les conditions d'applications du SLA (les clauses du contrat). Il peut s'agir typiquement de contraintes sur le volume de trafic.
- Le contrat de trafic (SLA+TCA) peut être implémenté en utilisant :
  - Un contrôle d'admission fondé sur le TCA,
  - Une réservation des ressources fondée sur le SLA.

Gestion de trafic
Qualité de service

- **Principales classes** de service (vision 3GPP) :
  - Classe conversationnelle : faibles délais et gigues. Exemple : voix.
  - Classe streaming : faible gigue. Exemple : flux vidéo.
  - Classe interactive : pas de pertes de données, réactivité requête-réponse. Exemple : navigation Internet.
  - Classe best effort : pas de contraintes fortes sur le délai mais pas de pertes de données. Exemple : mel, télémétrie.
- **Classification des flux** (vision télétrafic) :
  - Flux continus (*stream*) : audio, video, temps-réel, etc.
    - Caractéristiques : débit, durée, processus d'arrivée,
    - Critères de QoS : bande passante fixe, délai et gigue contrôlés.
  - Flux élastiques : non temps-réel, FTP, etc.
    - Le débit s'adapte à la bande (e.g. TCP),
    - Critères de QoS : temps de transfert, débit.

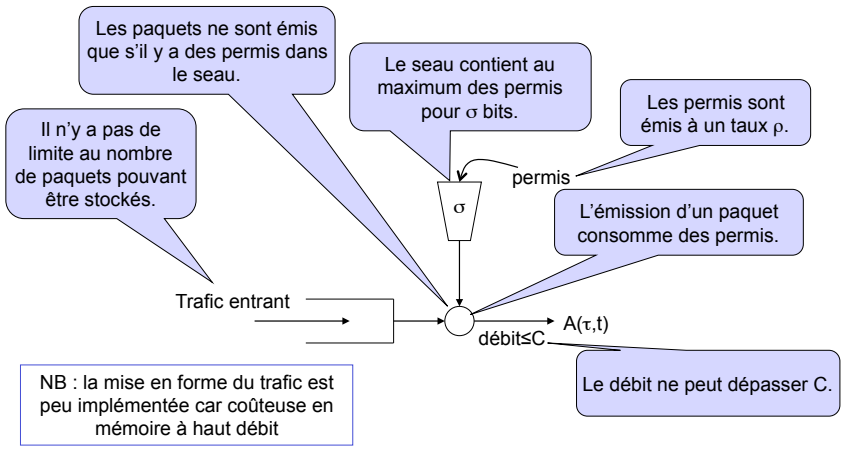
Gestion de trafic
Classification

- **Classification** des algorithmes de gestion du trafic :



Gestion de trafic
Seau percé

- Exemple de mise en forme/conditionnement du trafic : « le seau percé » (*leaky bucket*)



Gestion de trafic
Seau percé

- Si $A(t_1,t_2)$ est le volume de trafic qui entre dans le réseau après le seau percé entre t_1 et t_2 :

$$\forall (t_1,t_2),\ 0 \le t_1 \le t_2,\ A(t_1,t_2) \le \min\{(t_2-t_1)C, \sigma + \rho(t_2-t_1)\}$$

- Le débit moyen est contrôlé avec ρ, le débit pic avec C, la sporadicité avec σ et C. Le paramètre σ est la taille maximale d'une salve (*burst*).
- S'il n'y a pas de contrainte de débit maximum C (C=∞) :

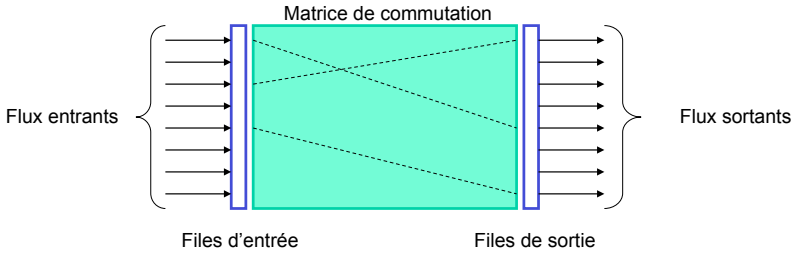
$$\forall (t_1,t_2),\ 0 \le t_1 \le t_2,\ A(t_2,t_1) \le \sigma + \rho(t_2-t_1)$$

- On dit que le trafic est contraint par un mécanisme de « *leaky bucket* ».

Gestion de trafic
Matrice de commutation

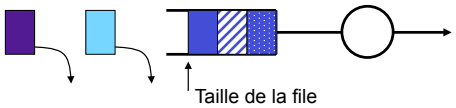
Principe simplifié d'un commutateur :

- Les paquets peuvent subir des délais aux niveaux :
  - Des files et de l'ordonnanceur d'entrée,
  - De la matrice de commutation (qui peut être bloquante),
  - Des files et de l'ordonnanceur de sortie.

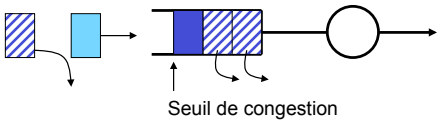


Gestion des files d'attente
Drop Tail et Selective Discard

- **But** de la gestion des files (*buffer management*) : sélectionner les paquets à supprimer en cas de congestion (la mémoire est pleine).
- **Drop Tail** : supprime les derniers paquets arrivés.
  - Implémentation très simple.
  - Problème : plusieurs flux TCP peuvent être impactés et vont réduire leur fenêtre d'émission (*slow start*) → oscillations possibles des débits.



- **Selective Discard** : seuls certains paquets (éligibles) sont supprimés.
  - Exemple : les paquets d'une connexion ou les cellules d'un paquet.

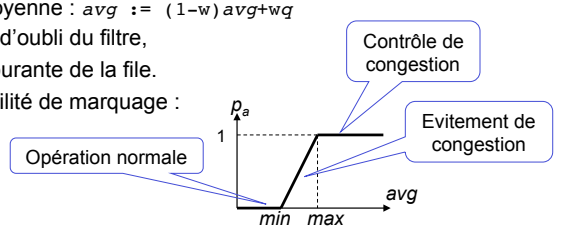


Gestion des files d'attente
RED

- **Random Early Detection** (RED) [7] :

```
A chaque arrivée de paquet
     Calcule la taille moyenne de la file avg
     Si min ≤ avg ≤ max
        Calcule la probabilité p_a
        Marque le paquet avec une probabilité p_a
     Sinon si max < avg
        Marque le paquet
```

- Estimation de la moyenne : $avg := (1-w)avg+wq$
  - w est le facteur d'oubli du filtre,
  - q est la taille courante de la file.
- Calcul de la probabilité de marquage :

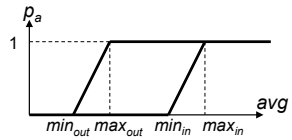


Propriétés de RED :

- Evitement de congestion : RED permet de contrôler la taille moyenne de la file (elle ne dépasse jamais max).
- Pas de synchronisation des sources : en faisant varier p_a, RED évite que l'ensemble des sources ne réduisent leur débit en même temps.
- L'implémentation est assez simple pour les routeurs de périphérie, mais inenvisageable pour les routeurs de coeur (➔ Drop Tail).
- Les simulations montrent que l'utilisation de la bande est supérieure qu'avec Drop Tail.
- Equité : les connexions sont affectées proportionnellement à la bande qu'elles occupent (comme Drop Tail).

Quelques variantes de RED :

- **RIO** (RED *with In-Profile and Out-Profile*) [8] : RED avec deux classes de paquets (les paramètres *min* et *max* sont définis par classe).



- **ARED** (*Adaptive* RED) [6] : les paramètres opérationnels sont adaptés de manière dynamique.
- **FRED** (*Flow Random Early Drop*) [9] : les paramètres sont maintenus par connexion de façon à obtenir une meilleure isolation entre flux (ex : une connexion UDP ne peut monopoliser la bande). Mais FRED ne passe pas à l'échelle (car fondée sur les flux).

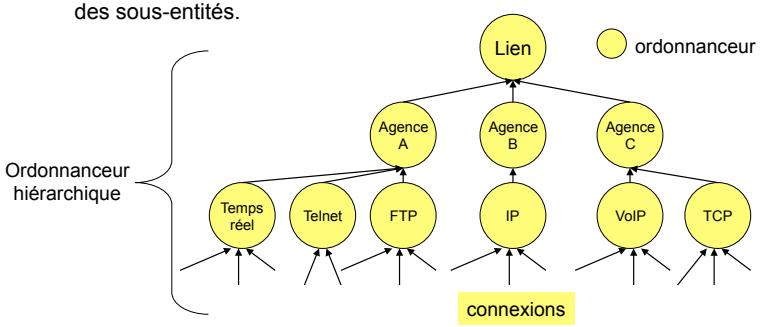
- **L'ordonnancement** détermine la manière dont sont partagées les ressources entre les flux actifs.
- Ressources = bande passante + temps + espace mémoire. L'attribution des ressources affecte les paramètres de performance des flux.
- Paramètres de performances : débit, délai et gigue, pertes.
- Ordonnancement = séquencement = discipline de service (en français).
- *Scheduling* = *service discipline* (en anglais).

Propriétés des algorithmes d'ordonnancement (*schedulers*) [1,10]:

- Isolation des flux : capacité à protéger les flux à service garanti de trois sources de variabilité :
  - Les flux qui émettent plus de paquets que ce qu'ils ont négocié,
  - Les fluctuations de la charge du réseau,
  - Le trafic au mieux (*best effort*) qui n'est pas contraint pas le contrôle d'admission.
- Equité (*fairness*) : capacité à allouer aux flux actifs les ressources de manière proportionnelle à leur besoins négociés.
- Passage à l'échelle (*scalability*) : capacité à supporter un grand nombre de flux et des débits importants.
- Simplicité : d'implémentation et d'analyse.

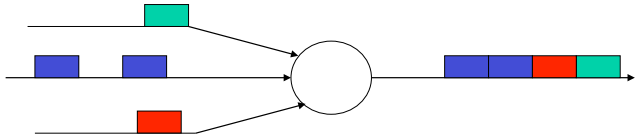
Ordonnancement
Définition et propriétés

- Différentiation de classes : capacité à pouvoir distinguer plusieurs classes de flux.
- Efficacité : un algorithme est plus efficace s'il permet une plus grande utilisation du réseau pour les mêmes garanties de performances.
- Ordonnanceur hiérarchique (vs. plat/*flat*) : les ressources sont partagées entre plusieurs entités et les ressources d'une entité sont récursivement allouées à des sous-entités.



Ordonnancement
Définition et propriétés

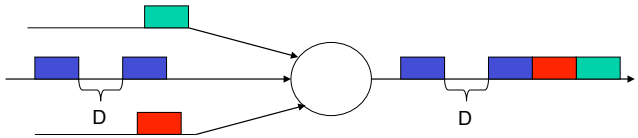
- Ordonnanceur sans oisiveté (*work-conserving*) : le serveur n'est jamais en veille s'il y a des données en attente.



- Avantage : bonne utilisation des ressources.
- Inconvénient : introduit des variations de délai importantes.

Ordonnancement
Définition et propriétés

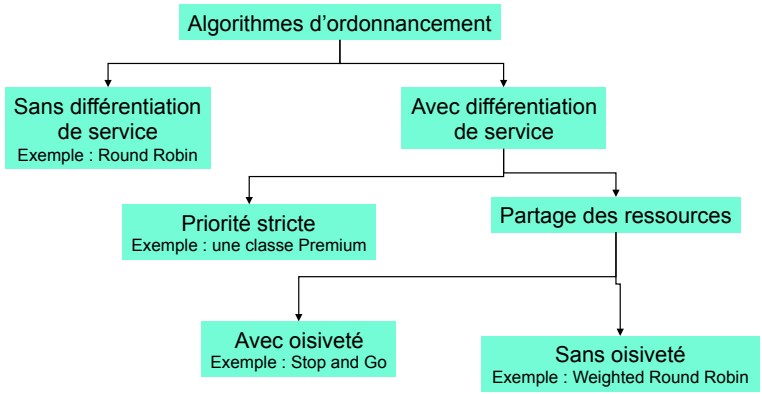
- Ordonnanceur avec oisiveté (*non-work-conserving*) :
  - Le contrôle du débit et l'ordonnancement sont découplés.
  - Le serveur peut être en veille alors que des paquets attendent.
  - On attribue au paquet une date d'éligibilité à laquelle il peut être transmis.



- Avantage : moins de variations de délai (gigue) ; trafic plus prédictible.
- Inconvénient : perte possible de ressources ; délai de bout en bout plus important ; difficile à implémenter en pratique.

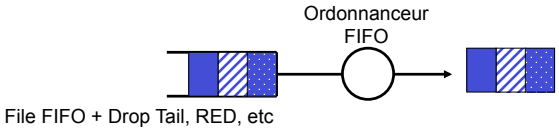
Ordonnancement
Classification

- Une classification possible des algorithmes [1] :



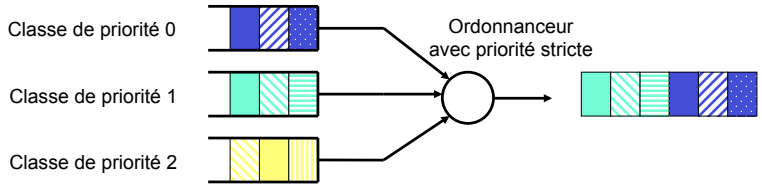
Ordonnancement
First In First Out

- FIFO est l'ordonnanceur le plus simple : les paquets sont servis dans l'ordre de leur arrivée.
- Caractéristiques :
  - Sans oisiveté,
  - Pas de différentiation de classe,
  - Plat,
  - Peut être couplé e.g. à Drop Tail (le plus commun) ou RED,
  - Simple ! (implémentation et analyse = M/G/1/K)
- Remarque : le terme FIFO s'applique aussi à des ordonnanceurs plus complexes e.g. en référence au traitement des paquets appartenant à une même classe.



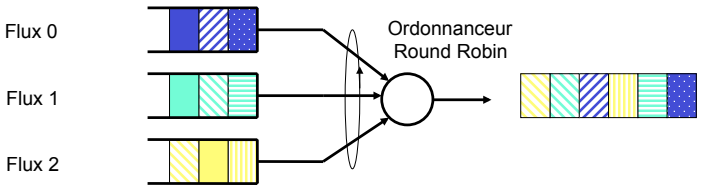
Ordonnancement
Priorité stricte

- On associe une file à chaque classe et on sert les classes de plus grande priorité en premier.
- Caractéristiques :
  - Plat et sans oisiveté,
  - Différentiation de classe,
  - Problèmes de **famine** (*starving*) : les classes de faible priorité peuvent ne jamais être servies,
  - On peut garantir un délai maximal pour la classe de plus grande priorité,
  - Il n'y a pas d'isolation des flux,
  - Il n'y a **pas d'équité**.



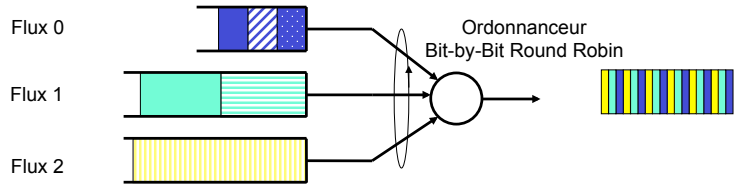
Ordonnancement
Round Robin

- Les flux sont servis à tour de rôle.
- Caractéristiques :
  - Plat,
  - Sans oisiveté : si une file est vide, on passe immédiatement à la suivante,
  - **Pas de différentiation de classe**,
  - Pas de cas de famine,
  - **Equité stricte** si les flux utilisent des paquets de même longueur,
  - Meilleure isolation des flux.



Ordonnancement
Bit-by-Bit Round Robin

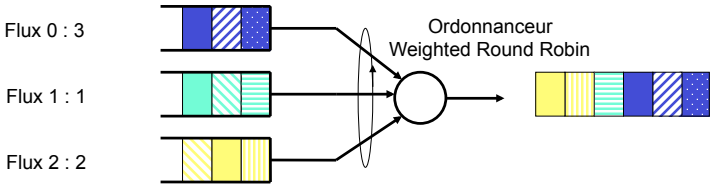
- BR [16-17] : algorithme Round Robin « parfait », c-a-d indépendant de la longueur des paquets et équitable sur (presque) toutes les échelles de temps.
- BR n'est qu'une **référence** et ne peut être implémenté dans un réseau à commutation de paquets.
- Des poids différents (ϕ_i) peuvent être attribués à des flux différents. Dans ce cas, plusieurs bits (ϕ_i) sont servis à chaque cycle.



Ordonnancement
Weighted Round Robin

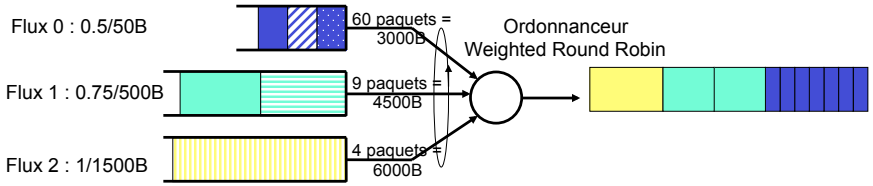
- WRR est la tentative la plus simple pour approcher BR dans un réseau à commutation de paquet.

- **Principe** :
  - Les poids sont normalisés par rapport à la taille moyenne des paquets,
  - Les poids sont à nouveau normalisés de façon à obtenir des entiers,
  - A chaque visite, l'ordonnanceur sert un nombre de paquets proportionnel au poids.



Ordonnancement
Weighted Round Robin

- Exemple : 3 flux avec des tailles moyennes de paquet différentes.
  - Tailles moyennes des paquets : {50 ; 500 ; 1500} octets,
  - Poids initiaux : {0.5 ; 0.75 ; 1},
  - Première normalisation : {0.5/50;0.75/500;1/1500}={60;9;4}/6000,
  - Seconde normalisation : {60 ; 9 ; 4}.



Ordonnancement
Weighted Round Robin

Limitations de WRR :

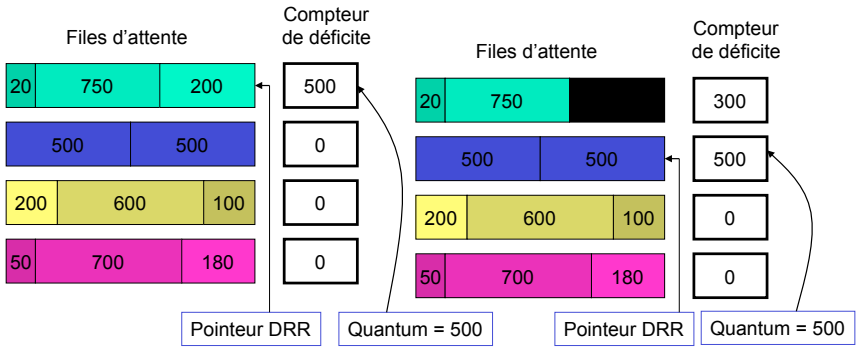
- La longueur moyenne des paquets doit être connue par flux (parfois difficile) ; si ce n'est pas le cas, l'ordonnancement n'est pas équitable. Remarque : en ATM, la taille des cellules est fixe.

- L'ordonnancement n'est équitable que sur une échelle de temps assez grande devant la durée du cycle.

- Les délais peuvent être assez longs surtout s'il y a de grandes différences entre les poids et si le nombre de flux est grand.

Ordonnancement
Deficit Round Robin

- Les auteurs de DRR [15] ont voulu réaliser un ordonnanceur avec une grande isolation entre flux, équitable, indépendant de la taille des paquets et surtout simple à implémenter.



- **Pseudo-code** pour DRR :

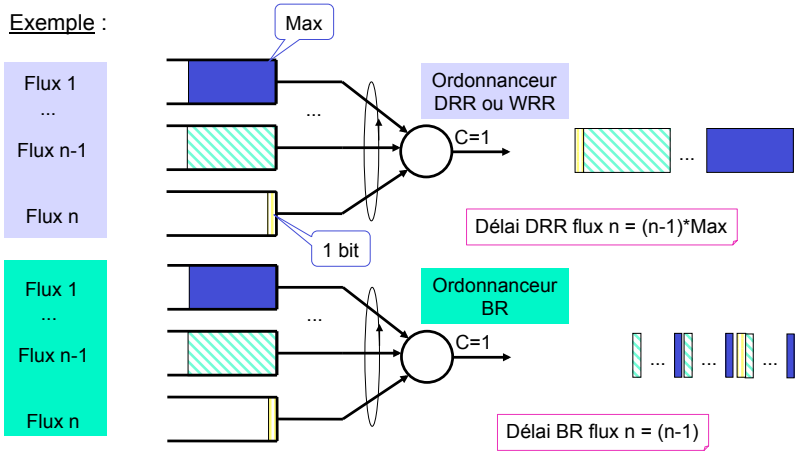
```
Si l'ActiveList n'est pas vide Alors
        Choisit le prochain flux i,
        DC_i := Q_i + DC_i
        Tant que DC_i>0 et que la file i n'est pas vide,
                Calcule la taille du premier paquet
                Si PacketSize ≤ DC_i, Alors
                        Emet le paquet
                        DC_i := DC_i - PacketSize
                Sinon sort de la boucle Tant que
        Si la file i est vide
                DC_i := 0
```

  - ActiveListe est la liste des flux actifs,
  - Q_i est le quantum associé au flux i,
  - $Q=Min_i(Q_i)$,
  - $\phi i = Q_i/Q$ est le poids associé au flux i,
  - Si la file est vide, le flux ne cumule pas de déficit.

- **Propriétés** de DRR (montrées dans [15], analyse et simulations) :
  - Très grande isolation des flux,
  - Coût d'implémentation assez faible (comparable à WRR),
  - Equitable quelque soit la taille des paquets,
  - Equitable pour différents types de trafic (Poisson et déterministe),
  - Mesure de l'équité (Max = longueur maximale d'un paquet) :

$$\forall(i,j),\ \forall(t_1,t_2),\ \left|\frac{W_i(t_1,t_2)}{\phi_i}-\frac{W_j(t_1,t_2)}{\phi_j}\right|\le 2Max+Q$$

- DRR ne résout pas le problème des longs délais qu'implique le cycle de l'ordonnanceur.
- Exemple :



- Virtual Clock s'inspire [12] des systèmes TDM (Time Division Multiplex).
- En **TDM**, chaque utilisateur utilise un slot, de sorte qu'il n'y a pas d'interférence entre les flux de données. En revanche, l'utilisation des ressources n'est pas optimal.
- **Virtual Clock** tente de reproduire ce comportement tout en bénéficiant du **gain statistique** inhérent à la commutation par paquets.

- Arrivée du premier paquet du flux *i* :
  - $VirtualClock_i := real\text{-}time$
- A chaque arrivée d'un paquet du flux *i* :
  - $auxVC_i := \max(real\text{-}time, auxVC_i)$,
  - $VirtualClock_i := VirtualClock_i + Vtick_i$,
  - $auxVC_i := auxVC_i + Vtick_i$,
  - marquer le paquet avec $auxVC_i$.
- Emettre les paquets dans l'ordre croissant de leur marque.
- S'il n'y a plus de place dans la mémoire, jeter le dernier paquet.

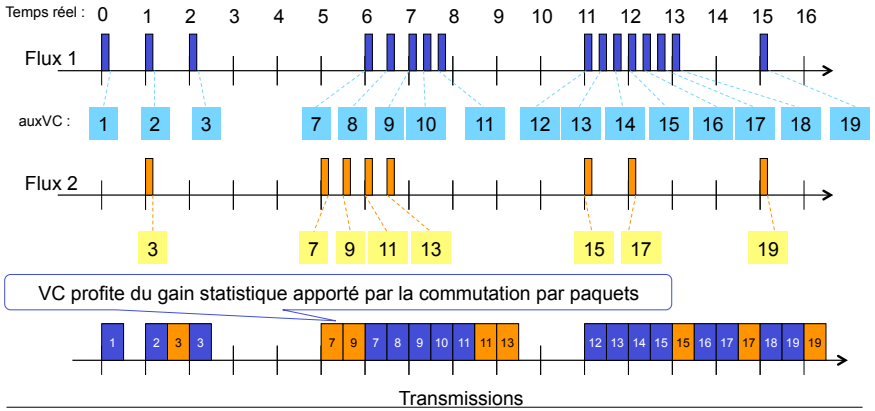
VirtualClock permet de contrôler le trafic du flux *i*

auxVC est la date à laquelle un système TDM aurait émis le paquet

Vtick est l'inverse du taux d'arrivée des paquets : $Vtick_i=1/AR_i$

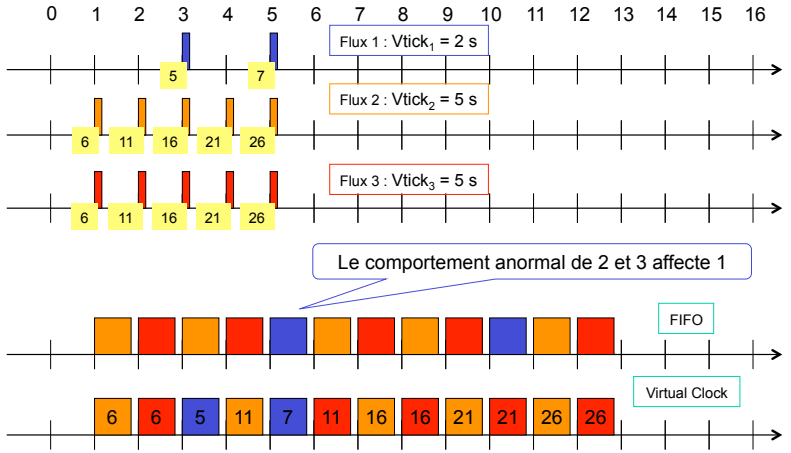
Ordonnancement
Virtual Clock

- **Exemple 1** : deux flux ont négocié à l'ouverture de leur connexion les débits respectifs AR_1 = 1 paquets/s et AR_2 = 0.5 paquets/s
- On en déduit : $Vtick_1$ = 1 s et $Vtick_2$ = 2 s



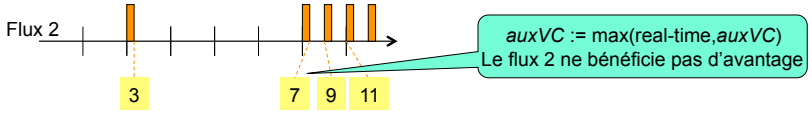
Ordonnancement
Virtual Clock

- **Exemple 2** : comparaison avec une file FIFO.



Ordonnancement
Virtual Clock

- **L'économie de bande n'est pas possible** : si un flux ne transmet pas de données pendant un certain temps,son horloge virtuelle est calée sur l'horloge réelle.



- **Contrôle de trafic** :
  - Si *VirtualClock* > real-time, alors le flux émet à un débit supérieur à celui prévu lors de l'établissement de la connexion.
  - Si *VirtualClock* < real-time, le flux émet à un débit inférieur.
  - *VirtualClock* est comparée à l'horloge réelle à intervalles réguliers.
  - En fait, il est assez difficile de conclure à un dépassement de débit (même avec des arrivées Poissonniennes) : les sources doivent contrôler leur débit pic.
  - En cas de dépassement, plusieurs mécanismes sont possibles : envoie d'un message de contrôle, transformation en flux best effort, suppression de paquets...

Ordonnancement
Virtual Clock

- Les **simulations** [12] montrent que :
  - Les débits demandés sont satisfaits, indépendamment du nombre de bonds (même à forte charge).
  - Le délai des paquets dans les files d'attente dépend peu du débit demandé (un flux à faible débit est servi moins souvent).
  - Les flux sont peu impactés par la sporadicité du trafic.
- Bons points :
  - on montre [13] que les délais sont bornés par auxVC+l^{max}/C, où
    - l^{max} est la taille maximale d'un paquet,
    - C est la capacité du lien de sortie.
  - VC est le premier algorithme à utiliser une horloge virtuelle.
- Mauvais point : VC n'est pas équitable [14].

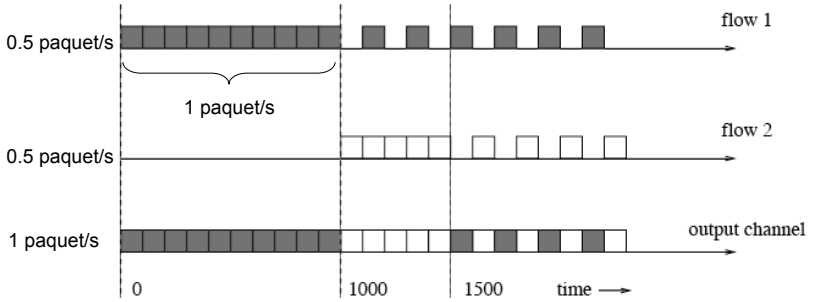
Qu'est-ce que l'équité (*fairness*) ?

- **Intuitivement**, un ordonnanceur est équitable s'il alloue la ressource disponible de manière proportionnelle au poids de chaque flux, et ce, quelque soit la période d'observation.

- **Définition** :
  - Soit ϕ_i le poids du flux i,
  - Soit $W_i(t_1,t_2)$ le volume de flux i servi entre t_1 et t_2,
  - Soit l_i^{max} la taille maximale d'un paquet du flux i.
  - L'allocation de ressource est équitable si :

$$\forall(i,j),\ \forall(t_1,t_2),\ \frac{W_i(t_1,t_2)}{\phi_i} = \frac{W_j(t_1,t_2)}{\phi_j}$$

- Exemple de période d'iniquité avec VC :



Dans [1000;1500] (t_1=1000, t_2=1500), le flux 1 ne reçoit pas de ressources alors que sa file n'est pas vide.

[14]

- **GPS** est une discipline de service parfaitement équitable [3-4]. C'est une généralisation de BR.
- **GPS** est un ordonnanceur sans oisiveté qui opère à un débit fixe *r*.
- Chaque flux est associé à un poids φ_i ,$1 \le i \le N$.
- Un ordonnanceur est GPS si pour tout flux *i* continuellement actif pendant [τ;t]:

Quantité de trafic *i* servi entre τ et t

$$\frac{W_i(\tau,t)}{W_j(\tau,t)} \ge \frac{\varphi_i}{\varphi_j}, j = 1,2,...,N$$

- En sommant sur tous les flux j :

$$W_i(\tau,t)\sum_j \varphi_j \ge (t-\tau) r \varphi_i$$

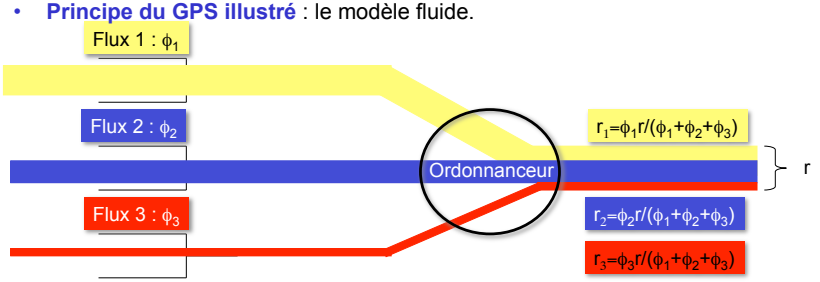
Débit garanti au flux *i* lorsque sa file n'est pas vide

$$g_i = \frac{\varphi_i}{\sum_j \varphi_j} r$$

- **Débit instantané** : Si dans l'intervalle [τ,t], l'ensemble des flux actifs, au nombre de B(τ), est fixe, le débit du flux *i* s'écrit :

$$r_i(\tau,t) = \frac{\phi_i}{\sum_{j\in B(\tau)} \phi_j} r$$

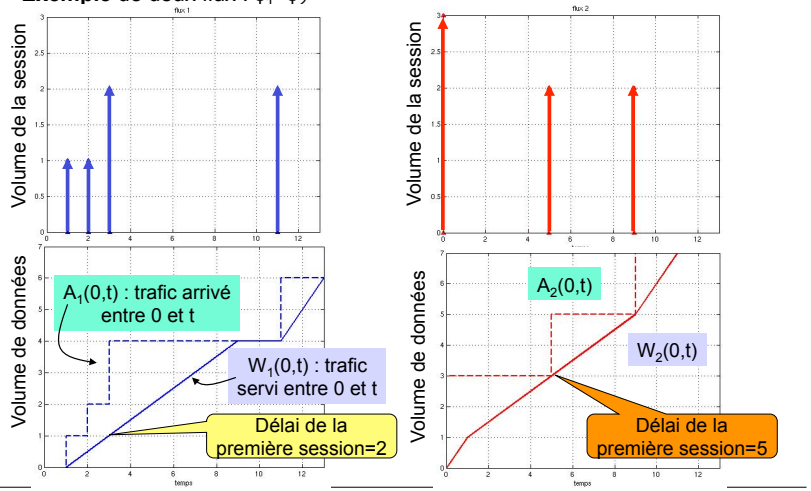
- **Principe du GPS illustré** : le modèle fluide.



- **Propriétés** :
  - Si le débit moyen du flux i est ≤ g_i, on peut garantir au flux *i* un débit indépendant des autres flux.
  - Le délai d'une session du flux *i* peut être borné par une fonction de la longueur de la file *i* indépendamment des files des autres flux.
  - Les poids offrent une grande flexibilité dans le traitement des flux.
  - Lorsque le trafic est contraint par un Leaky Bucket (σ,ρ), il est possible de garantir un délai dans la file d'attente.

- **Inconvénients** :
  - GPS ne considère que des flux et non des paquets en tant qu'entités,
  - L'ordonnanceur est supposé servir plusieurs flux simultanément,
  - Le trafic est supposé indéfiniment divisible,
  - Problème : l'implémentation.

- **Exemple** de deux flux : $\phi_1=\phi_2$

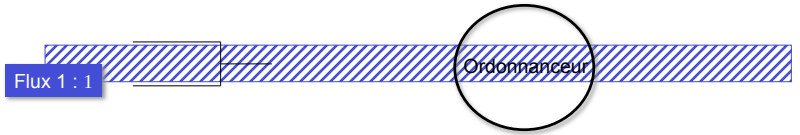


- Entre 0 et 1 :

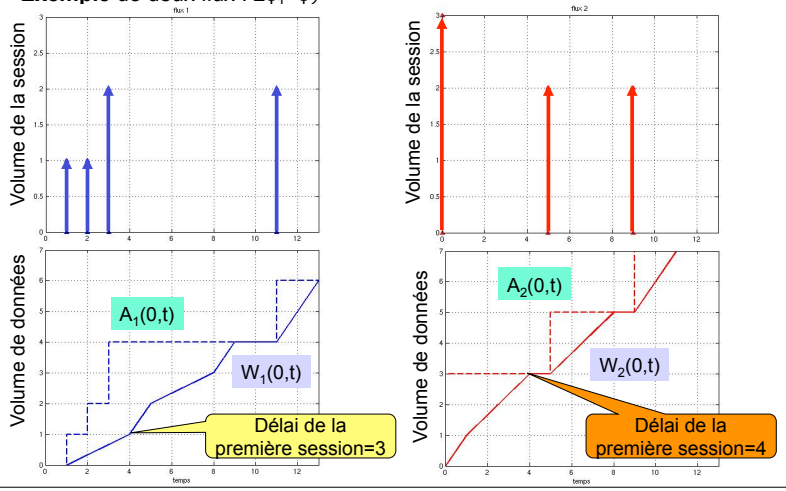
- Entre 1 et 9 :

- Entre 9 et 11 :

- Entre 11 et 13 :



- **Exemple** de deux flux : $2\phi_1=\phi_2$



- **Borne sur le délai** :
  - Les sources sont contraintes par un leaky bucket (σ_i,ρ_i).
  - Le système est stable ssi $\rho_i \le g_i$.
  - Soit q_i^* la taille maximale de la file d'attente du flux *i*.
  - Soit d_i^* le délai d'attente maximal d'un bit du flux *i*.
  - Supposons que q_i^* soit atteinte au temps t. Soit τ, le premier instant avant t auquel la file d'attente du flux *i* est vide.

$$\left.\begin{aligned} W_i(\tau,t) &\ge (t-\tau)g_i \ge (t-\tau)\rho_i \\ A_i(\tau,t) &\le \sigma_i + \rho_i(t-\tau) \\ q_i^* &= A_i(\tau,t) - W_i(\tau,t) \end{aligned}\right\} \Rightarrow q_i^* \le \sigma_i$$

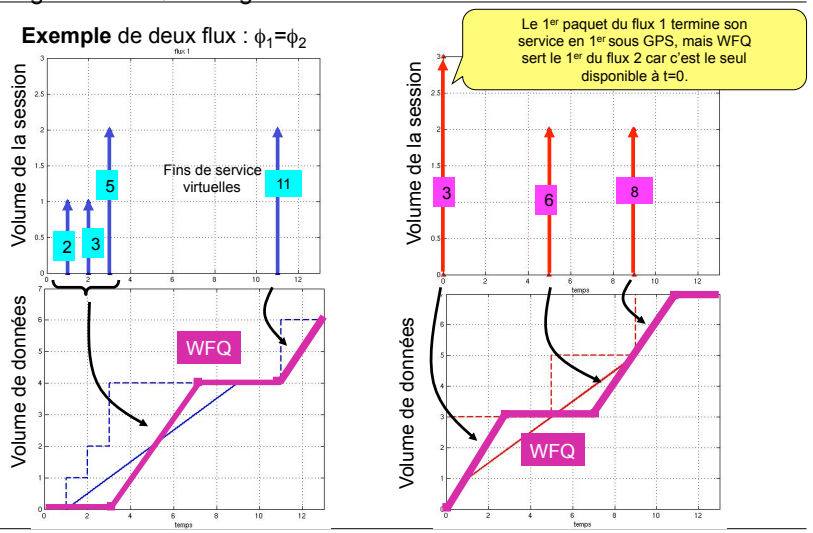
  - Un bit du flux *i* sera servi après au plus q_i^* bits. Ces bits seront servis à un débit au moins égal à g_i. Donc $d_i^* \le q_i^* / g_i$.

$$d_i^* \le \frac{\sigma_i}{g_i}$$

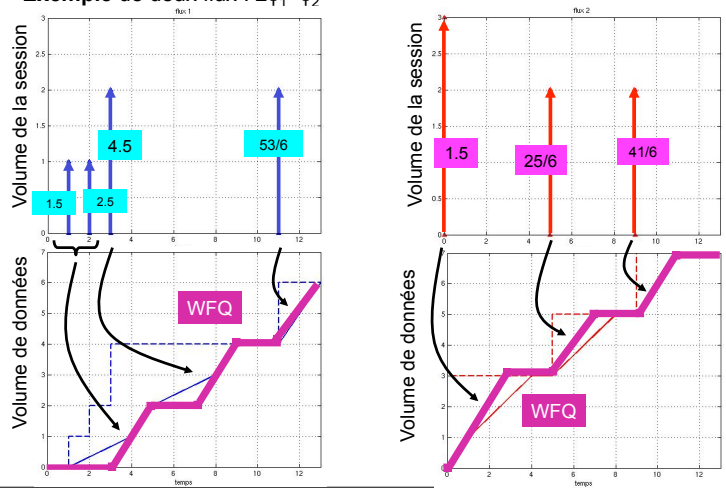
Définition

- Dans un réseau de **paquets**, une seule connexion peut être servie à la fois et un paquet doit être transmis entièrement avant qu'un autre puisse l'être.
- **WFQ** (ou PGPS pour Packet-by-packet GPS) est une approximation de GPS pour les réseaux de paquets. WFQ est sans oisiveté.
- Soit F_p la date à laquelle le paquet p finit son service avec GPS. A un instant donné, WFQ ne peut pas servir les paquets par ordre croissant de F_p car certains paquets ne sont pas encore arrivés.
- Discipline de service WFQ : a un instant t, WFQ classe les paquets par ordre croissant de F_p et choisit le paquet qui aurait le premier fini sont service sous GPS **s'il n'y avait aucune arrivée après t**.

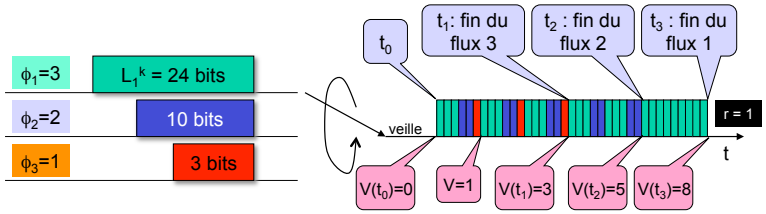
- **Exemple** de deux flux : $\phi_1=\phi_2$



- **Exemple** de deux flux : $2\phi_1=\phi_2$



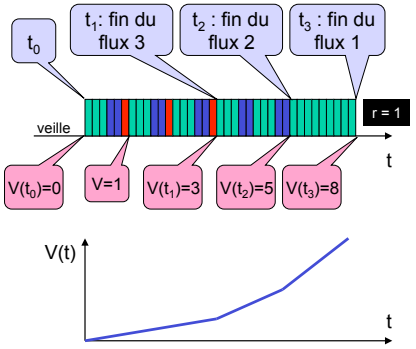
- Implémentation avec **horloge virtuelle**.
- On émule GPS par un Round Robin qui servirait tour à tour une partie infinitésimale de chaque flux (par exemple bit à bit) :



- L'horloge virtuelle est incrémentée de 1 à chaque tour de l'ordonnanceur. Ceci permet d'exprimer simplement le temps de service d'un paquet indépendamment des autres flux.
  - Exemple : fin de service du paquet du flux 1 = $V(t_3) = L_1^k/\phi_1 = 8$

- L'horloge virtuelle est incrémentée de 1 à chaque tour de l'ordonnanceur.
- L'horloge virtuelle est nulle lorsque l'ordonnanceur est **inactif**.
- Pendant les périodes d'activité, c'est une **fonction linéaire par morceaux** dont la pente dépend du nombre de flux actifs.
- La pente est remise à jour à chaque événement, c.a.d. arrivée ou départ de paquet du GPS. Le j^{ième} **événement** se produit au temps t_j.
- Entre t_{j-1} et t_j, l'ensemble des flux actifs, B_j, est fixe.

$$\begin{cases} V(0)=0 \\ \forall \tau \le t_j - t_{j-1}, V(t_{j-1}+\tau) = V(t_{j-1}) + \dfrac{\tau}{\sum_{i\in B_j}\phi_i} \end{cases}$$



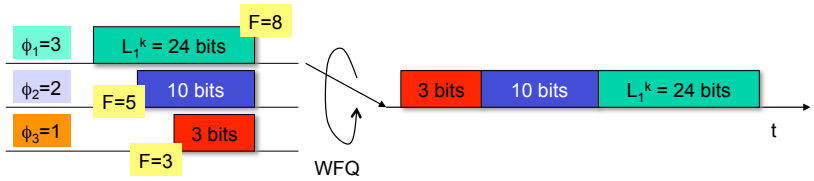
- Soit S_i^k et F_i^k les temps virtuels de début et de fin de service du $k^{ième}$ paquet du flux *i* arrivé à $t = a_i^k$:
  - Si la file *i* est vide à son arrivée : $S_i^k = V(a_i^k)$.
  - Si la file *i* n'est pas vide à son arrivée : $S_i^k = F_i^{k-1}$ (on fixe $F_i^0 = 0$).
  - Et $F_i^k = S_i^k + L_i^k/\phi_i$.

$$\begin{cases} S_i^k = \max\left\{F_i^{k-1}, V(a_i^k)\right\} \\ F_i^k = S_i^k + \dfrac{L_i^k}{\phi_i} \\ F_i^0 = 0 \end{cases}$$

- **Pseudo-code** :

```
• A l'arrivée d'un paquet (k^ième du flux i) à t :
    • Mettre à jour de V(t)
    • Calculer Fik
    • Marquer le paquet avec Fik

• En fin de service d'un paquet :
    • Choisir le paquet avec la marque la plus petite
```



- **Comparaison avec GPS** :
  - Si on considère l'ensemble des paquets en attente à un instant t, **l'ordre de fin de service** de ces paquets est le même pour GPS et WFQ.
  - Soient F_p et F'_p les dates de fin de service de p avec resp. GPS et WFQ et L_{max} la taille maximale d'un paquet :

$$F'_p - F_p \leq L_{max} / r$$

  - Soient $W_i(t)$ et $W'_i(t)$ le volume de trafic écoulé pour le flux *i* entre 0 et t avec resp. GPS et WFQ :

$$W_i(t) - W'_i(t) \leq L_{max}$$

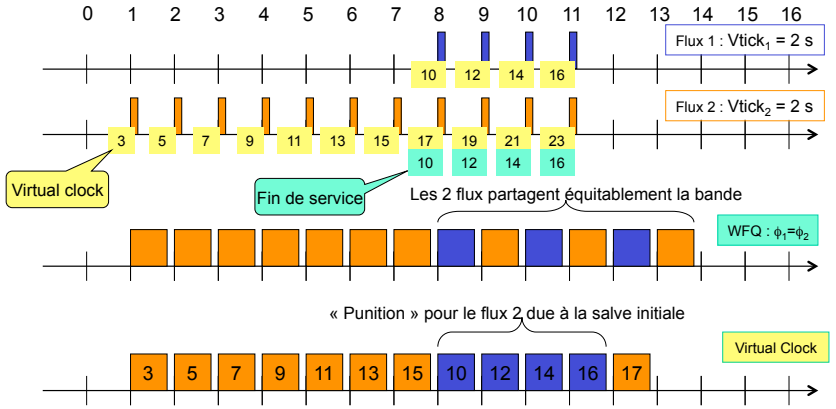
  - Soient $Q_i(t)$ et $Q'_i(t)$ le volume de trafic en attente pour le flux *i* à t avec resp. GPS et WFQ :

$$Q'_i(t) - Q_i(t) \leq L_{max}$$

  - Conclusions : les bornes obtenues avec GPS peuvent être transcrites pour WFQ et WFQ n'est jamais très en retard sur GPS.

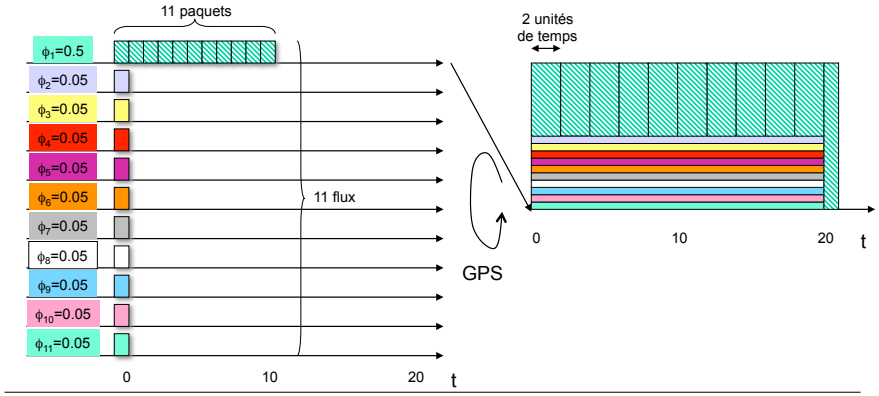
- **Comparaison avec WRR** :
  - WRR ne prend pas suffisamment en compte les tailles variables de paquets.
- **Comparaison avec Virtual Clock** :
  - Virtual Clock et WFQ utilisent tous deux un marquage temporel pour déterminer l'ordre de service. Virtual Clock se fonde sur un système TDM, alors que WFQ se fonde sur un ordonnanceur GPS.
  - Avec Virtual Clock, un flux peut être « puni » en cas d'émission d'un burst, même si celui-ci n'a pas affecté les autres flux (cf. transparent suivant).

- **Exemple** : comparaison WFQ / Virtual Clock.



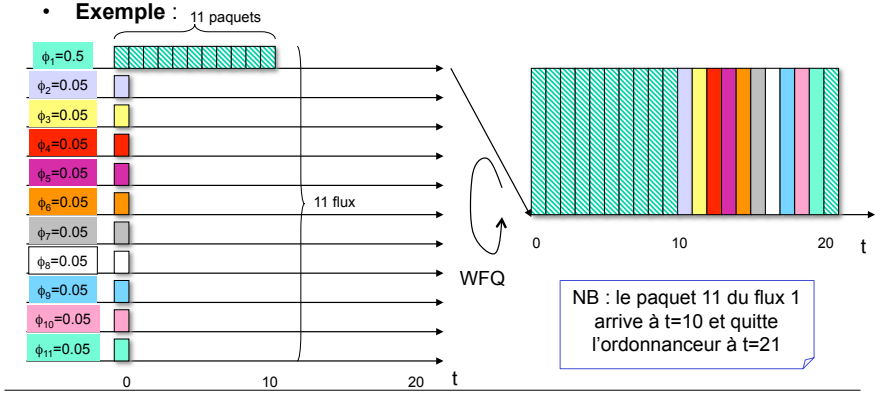
Ordonnancement
Weighted Fair Queueing

- La comparaison de WFQ et GPS montre que WFQ n'est jamais très en retard sur GPS (à la longueur d'un paquet près). Il se peut en revanche que WFQ soit bien **en avance** sur GPS.
- **Exemple** :



Ordonnancement
Weighted Fair Queueing

- Les dix premiers paquets du flux 1 terminent leur service bien avant avec WFQ qu'avec GPS. Le flux 1 peut ainsi alterner périodes de fort et de faible débit.
- Ces oscillations sont néfastes pour le contrôle de congestion ou de flux.
- **Exemple** :



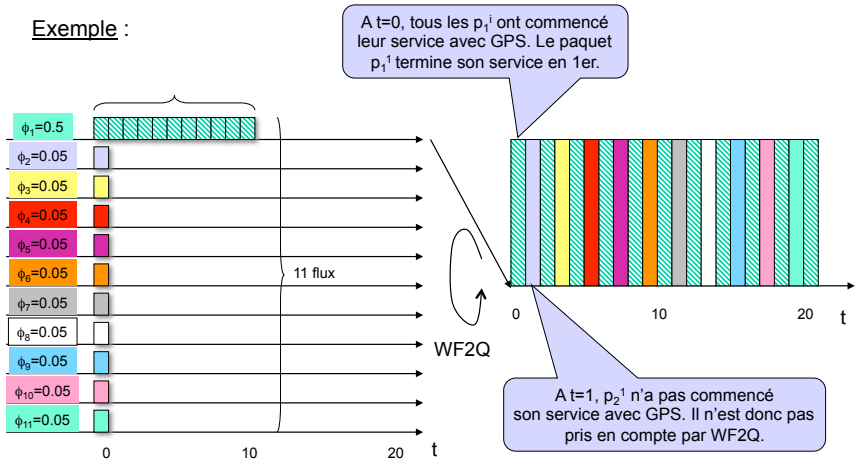
Ordonnancement
Worst Case Fair Weighted Fair Queueing

Définition

- A la différence de WFQ, WF2Q ne considère que les paquets qui ont commencé leur service avec GPS.
- WF2Q considère donc à la fois le **début** et la **fin de service** avec GPS.
- Le but est de réduire les oscillations observées avec WFQ et d'éviter que l'ordonnanceur ne soit trop en avance sur GPS.
- Discipline de service WF2Q : a un instant t, WF2Q choisit parmi les paquets qui ont **commencé** leur service avec GPS, le paquet qui aurait le premier fini sont service sous GPS **s'il n'y avait aucune arrivée après t**.

Ordonnancement
Worst Case Fair Weighted Fair Queueing

Exemple :



Ordonnancement
Worst Case Fair Weighted Fair Queueing

- Soient F_p et F'_p les dates de fin de service de p avec resp. GPS et WF2Q et L_{max} la taille maximale d'un paquet :

$$F'_p - F_p \leq L_{max} / r$$

- Soient $W_i(t)$ et $W'_i(t)$ le volume de trafic écoulé pour le flux *i* entre 0 et t avec resp. GPS et WF2Q et g_i le débit garanti pour le flux *i* :

$$W_i(t) - W'_i(t) \leq L_{max}$$

$$W'_i(t) - W_i(t) \leq (1-g_i/r)L_{max}$$

- Cette dernière relation montre que WF2Q n'est jamais très en avance, ni très en retard par rapport à GPS, ce qui n'était pas garanti par WFQ.

Implémentations

- Les implémentations pour produits dépendent de plusieurs facteurs [1] :
  - La bande : généralement les algorithmes complexes ne sont pas implémentés pour de grands débits.
  - Les ordonnanceurs fondés sur les flux peuvent devenir très complexes lorsque le nombre de flux augmente ➔ pas plus de quelques Mbps. Plus il y a de files d'attentes différentes, plus il faut de mémoire.
  - Les ordonnanceurs fondés sur les classes sont plus adaptés aux gros débits (les classes sont moins nombreuses que les flux).

Implémentations

- **Routeurs de périphérie** (*edge device*) :
  - Priorité stricte pour une file avec faible délai d'attente,
  - WFQ entre différentes classes de trafic,
  - FIFO à l'intérieur d'une classe entre flux,
  - Une classe peut parfois être servie avec un WFQ fondé sur les flux,
  - Nombre typique de files : ~64 (63+1 avec faible délai - *low-latency queue*).
- **Routeurs de coeur** (*core device*) :
  - Priorité stricte pour une file avec faible délai,
  - DRR ou WRR (plus rarement WFQ) entre différentes classes de trafic,
  - FIFO à l'intérieur de chaque classe entre flux,
  - Nombre typique de files : ~8 (7+1).
- Rappel : les produits se distinguent aussi par leur matrice de commutation (*switching fabric*).

Conclusion

- Gestion des files d'attente : en pratique, seule Drop Tail est utilisée. RED et a fortiori RIO sont trop complexes à haut débit.
- Ordonnancement :
  - Les variantes de RR sont simples ➔ routeurs de coeur,
  - Les ordonnanceurs équitables sont complexes ➔ routeurs de périphérie,
  - Types d'ordonnanceurs non abordés dans ce cours :
    - Stochastic Fair Queueing,
    - Les ordonnanceurs hiérarchiques (e.g. CBQ),
    - Les ordonnanceurs avec oisiveté (e.g. Stop and Go),
    - Les ordonnanceurs pour réseaux radio (max C/I, *proportional fair*,...).

Glossaire

3GPP : 3rd Generation Partnership Project
AR : Arrival Rate
ARED : Adaptive RED
BR : Bit-by-bit Round Robin
CBQ : Class Based Queueing
DRR : Deficit Round Robin
FIFO : First In First Out
FRED : Flow Random Early Drop
FTP : File Transfer Protocol
GPS : Generalized Processor Sharing
IP : Internet Protocol
PGPS : Packet-by-Packet GPS
QoS : Qualité de Service
RED : Random Early Discard
RIO : RED with In-Profile and Out-Profile
RR : Round Robin
SLA : Service Level Agreement
TCA : Trafic Conditioning Agreement
TCP : Transport Control Protocol
TDM : Time Division Multiplex
UDP : User Datagram Protocol
VC : Virtual Clock
VoIP : Voice over IP
WF2Q : Worst Case Fair Weighted Fair Queueing
WFQ : Weighted Fair Queueing
WRR : Weighted Round Robin

Références

[1] R. Casellas, G. Hébuterne, D. Kofman, M. Marot, J.L. Rougier, « Scheduling and Switching Architecture », ENST, rapport interne, 2004.
[2] A. Demers, « Analysis and Simulation of a Fair Queueing Algorithm »
[3] A. K. Parekh et R. G. Gallager, « A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single-Node Case », IEEE/ACM Trans. On Networking, June 1993.
[4] A. K. Parekh et R. G. Gallager, « A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Multiple Node Case », IEEE/ACM Trans. On Networking, April 1994.
[5] J. C. R. Bennett et H. Zhang, « Worst-case Fair Weighted Fair Queueing », 1996.
[6] S. Ryu et al., « Advances in Internet Congestion Control », IEEE Communications Surveys and Turorial, 3rd Quarter 2003.
[7] S. Floyd and V. Jacobson, « Random Early Detection Gateways for Congestion Avoidance », IEEE Trans. On Networking, Aug. 1993.
[8] D. D. Clark, « Explicit Allocation of Best-Effort Packet Delivery Service », IEEE Trans. On Networking, Aug. 1998.
[9] D. Lin and R. Morris, « Dynamics of Random Early Detection », ACM SIGCOMM, Oct. 1997.
[10] H. Zhang, « Service Disciplines for Guaranteed Performance Service in Packet-Switching Networks », Proceedings of the IEEE, Oct. 1995.
[11] S. Floyd et V. Jacobson, « Link Sharing and Resource Management Models for Packet Networks », IEEE/ACM Trans. On Networking, Aug. 1995.
[12] L. Zhang, « Virtual Clock: a New Traffic Control Algorithm for Packet Switching Networks », ACM SIGCOMM, Sept. 1990.
[13] G. Xie et S. Lam, « Delay Guarantee of Virtual Clock Server », IEEE/ACM Trans. On Networking, Dec. 1995.
[14] S. Suri et G. Varghese, « Leap Forward Virtual Clock », Rapport Washington University, 1996.
[15] M. Shreedhar et G. Varghese, « Efficient Fair Queuing Using Deficit Round Robin », IEEE/ACM Tans. On Networking, June 1996.
[16] A. Demers et S. Shenker, « Analysis and Simulation of a Fair Queueing Algorithm », ACM SIGCOMM, 1989.
[17] J. Nagle, « On Packet Switches with Infinite Storage », IEEE Trans. On Communications, 1987.

Licence de droits d'usage