Introduction au calcul réticulaire

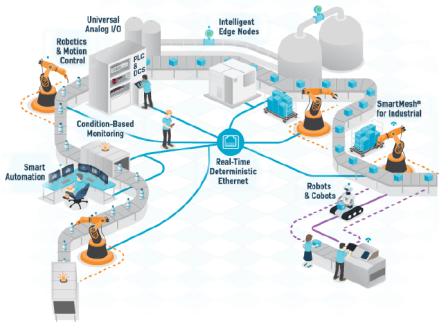
Le mot réticulaire vient du latin reticulum, qui signifie *petit filet* ou *réseau*.

Anne Bouillard, ENS

École d'été Temps Réel, le 11 septembre 2026

Réseaux sensibles au temps

Les réseaux sensibles au temps (Time-sensitive networks, TSN) doivent satisfaire des contraintes de délai de bout-en-bout pour assurer un fonctionnement correct.



Source : Analog Devices Inc. Industry 4.0

- **Objectif :** des garanties temporelles dans les réseaux
- **Domaines :** avionique, réseaux véhiculaires embarqués, réseaux industriels
- **Des réseaux hétérogènes :**
  - Classes de trafic : contrôle, capteurs, trafic vidéo (assistance de taches), images (contrôle de qualité).
  - Connections: filaires (Ethernet), sans fil.

Objectif : des bornes de performance pire-cas

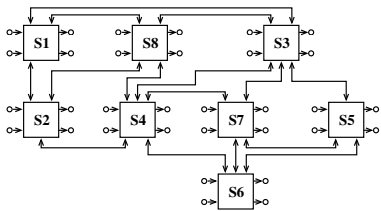
Calculer le temps de transmission maximum d'un paquet pour traverser un système (délai pire-cas (de bout-en-bout))

Calcul réticulaire (Network Calculus)

- Théorie développée dans les années 1990 par R.L. Cruz, puis développée et popularisée par C.S. Chang et J.-Y. Le Boudec.
- Une théorie de filtrage dans l'algèbre (min,plus).
- Applications:
  - Internet : transmission vidéo (VoD),
  - Équilibrage de charge dans les routeurs [Birkhoff-von Neumann switches, C.S. Chang]
  - Réseaux embarqués: AFDX (Avionics Full Duplex) [Logiciel Rockwell-Collins pour certifier l'A380], Networks-on-chip
- Extensions / variations:
  - Calcul Temps-réel [L. Thiele, S. Chakraborty]
  - Étendu au calcul réticulaire stochastique [C.S. Chang, Y.M. Jiang, F. Ciucu, J. Schmitt]
- A été plus récemment utilisé pour les réseaux 5G, qui ont des fortes contraintes de délais et fiabilité.

Réseaux complexes

Exemple d'une topologie de réseau industriel



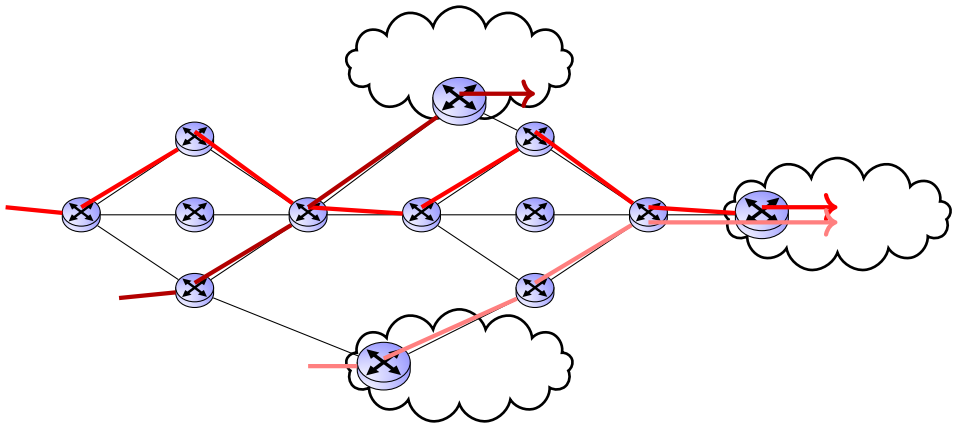
- 96 terminaux, 8 routeurs, 984 flots
- taux de service: 1 Gb/s; latence du routeur: 16μs.
- Trois classes of flots: (1) critique, (2) multimédia et (3) "au mieux" (best-effort).

class	1	2	3
flows	128	500	266

Caractéristiques du réseau :

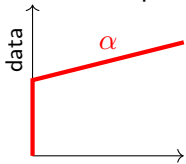
- **Multi-classe :** Chaque flot appartient à une classe de trafic, et partage la bande passante avec les autres classes.
- **Dépendances cycliques:** Les flots créent des dépendances cycliques, rendant l'analyse plus complexe.

Calcul réticulaire (Network Calculus)

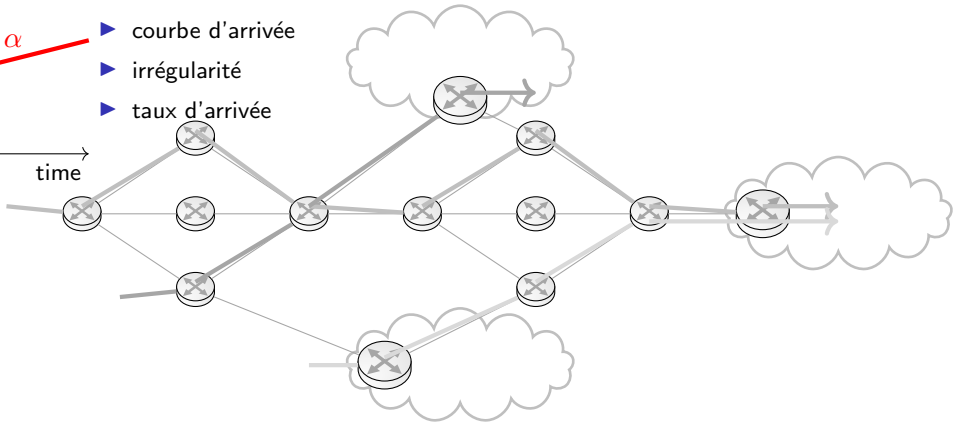


Calcul réticulaire (Network Calculus)

1. Modèle pour les données

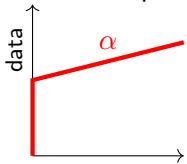


- courbe d'arrivée
- irrégularité
- taux d'arrivée



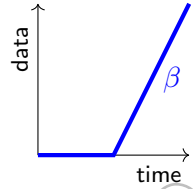
Calcul réticulaire (Network Calculus)

1. Modèle pour les données



- courbe d'arrivée
- irrégularité
- taux d'arrivée

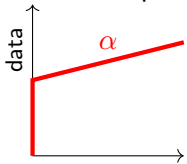
2. Modèle pour les serveurs



- courbe de service
- ordonnancement
- bande passante

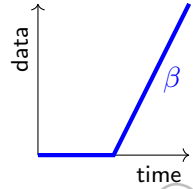
Calcul réticulaire (Network Calculus)

1. Modèle pour les données



- courbe d'arrivée
- irrégularité
- taux d'arrivée

2. Modèle pour les serveurs



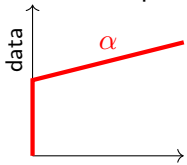
- courbe de service
- ordonnancement
- bande passante

3. Analyse des performances

- délais pire-cas
- de bout en bout

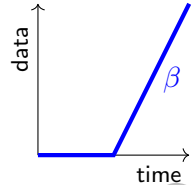
Calcul réticulaire (Network Calculus)

1. Modèle pour les données



- courbe d'arrivée
- irrégularité
- taux d'arrivée

2. Modèle pour les serveurs



- courbe de service
- ordonnancement
- bande passante

3. Analyse des performances

- délais pire-cas
- de bout en bout

4. Optimisation et conception

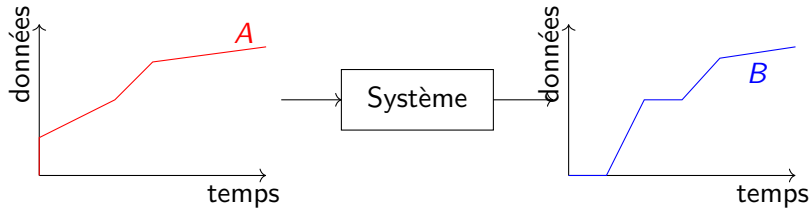
- optimisation des paramètres
- routage
- contrôle d'admission

Contents

Calcul réticulaire : un outil de modèlisation des réseaux

Calcul réticulaire : des outils d'analyse de performance pour les réseaux
- Pour géré un flot traversant plusieurs serveurs
- Pour gérer plusieurs classes de service·
- Analyse dans les réseaux complexes
- Dépendances cycliques

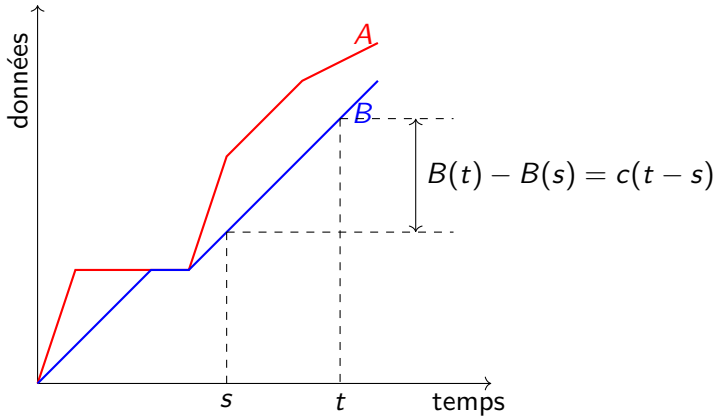
Processus cumulés



- $A : \mathbb{R}_+ \to \mathbb{R}_{\min +}$: processus des arrivées cumulées, fonction croissante
- $B : \mathbb{R}_+ \to \mathbb{R}_{\min +}$: process des départs cumulés, fonction croissante
- Contrainte de causalité : $A \geq B$

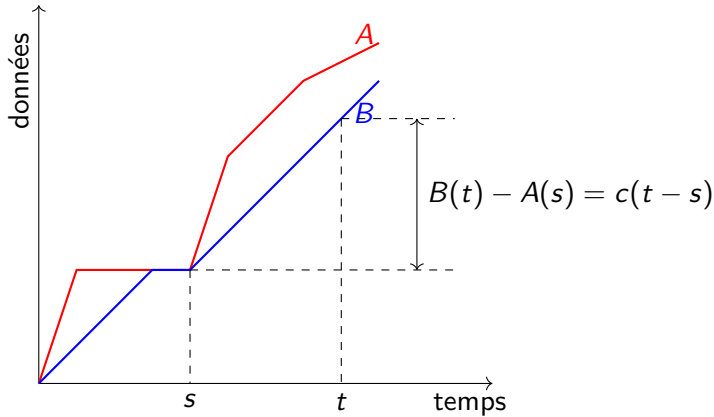
Exemple d'un serveur à taux constant

Supposons que le système sert exactement c bits de données par unité de temps.



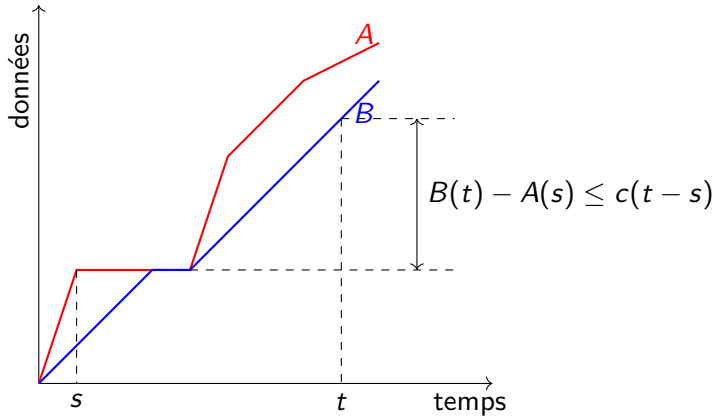
Exemple d'un serveur à taux constant

Supposons que le système sert exactement c bits de données par unité de temps.



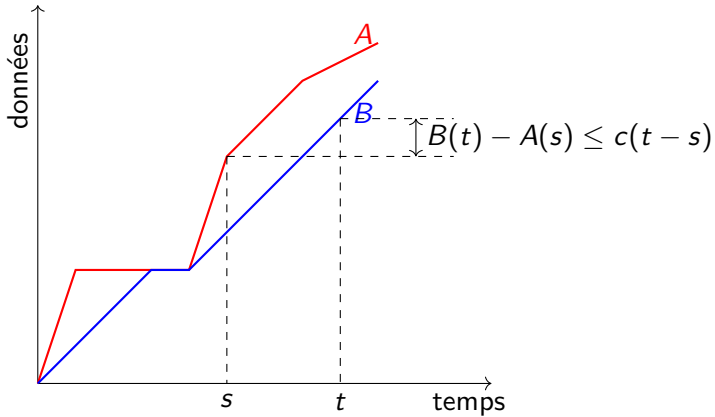
Exemple d'un serveur à taux constant

Supposons que le système sert exactement c bits de données par unité de temps.



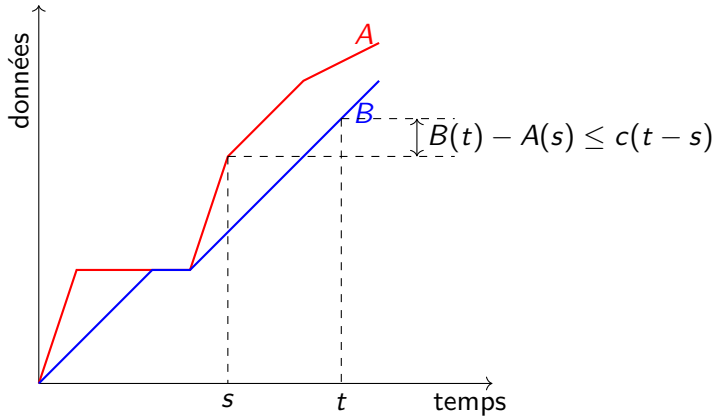
Exemple d'un serveur à taux constant

Supposons que le système sert exactement c bits de données par unité de temps.



Exemple d'un serveur à taux constant

Supposons que le système sert exactement c bits de données par unité de temps.



$$B(t) = \inf_{0 \leq s \leq t} A(s) + c(t-s) = A * c(t)$$

Une théorie basée sur l'algèbre (min,plus)

Semi-anneau (min,plus)$(\mathbb{R}_{\min}, \oplus, \otimes)$

$\mathbb{R}_{\min} = \mathbb{R} \cup \{+\infty\}$ avec deux opérateurs, $\oplus = \min$, $\otimes = +$.

- Associativité et commutativité de $\oplus$ et $\otimes$, distributivité $(a \oplus b) \otimes c = a \otimes c \oplus b \otimes c$;
- $a \otimes +\infty = a$, $a \otimes +\infty = \infty$ et $a \times 0 = 0$ (∞ : élément nul, 0 : élément unité) ;
- pas un anneau : $\oplus$ n'est pas inversible ;
- idempotent : $a \oplus a = a$.

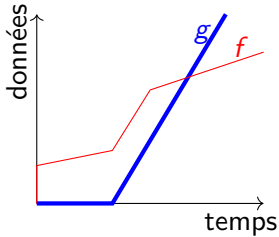
Semi-anneau des fonctions $(\mathbb{R}_+ \to \mathbb{R}_{\min,+}, \oplus, *)$

- $f \oplus g = \min(f, g)$ (point à point, classique)
- convolution (min,plus) $\forall t \in \mathbb{R}_+,\ f * g(t) = \inf_{0 \le s \le t} f(s) + g(t-s)$
- deconvolution (min,plus) $\forall t \in \mathbb{R}_+,\ f \oslash g(t) = \sup_{s \ge 0} f(t+s) - g(s)$ (pseudo-inverse)

Calcul de l'opérateur $*$

Convolution (min,plus) $f * g(t) = \inf_{0 \leq s \leq t} f(s) + g(t-s)$

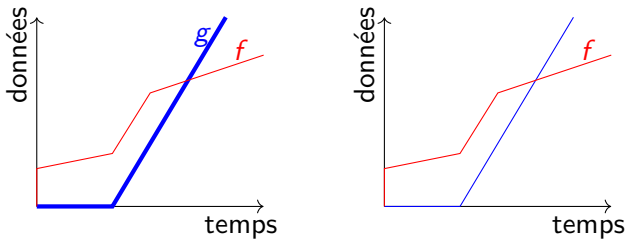
On glisse g sur f et on prend l'enveloppe minimale



Calcul de l'opérateur $*$

Convolution (min,plus) $f * g(t) = \inf_{0 \leq s \leq t} f(s) + g(t-s)$

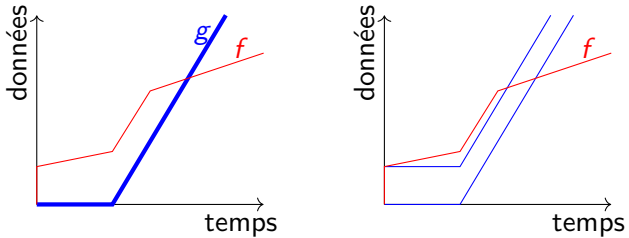
On glisse g sur f et on prend l'enveloppe minimale



Calcul de l'opérateur $*$

Convolution (min,plus) $f * g(t) = \inf_{0 \leq s \leq t} f(s) + g(t-s)$

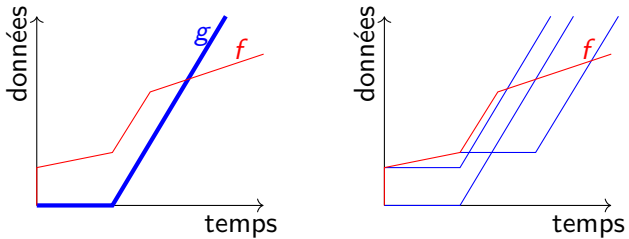
On glisse g sur f et on prend l'enveloppe minimale



Calcul de l'opérateur $*$

Convolution (min,plus) $f * g(t) = \inf_{0 \leq s \leq t} f(s) + g(t-s)$

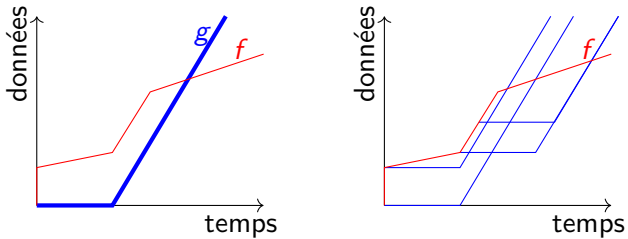
On glisse g sur f et on prend l'enveloppe minimale



Calcul de l'opérateur $*$

Convolution (min,plus) $f * g(t) = \inf_{0 \leq s \leq t} f(s) + g(t-s)$

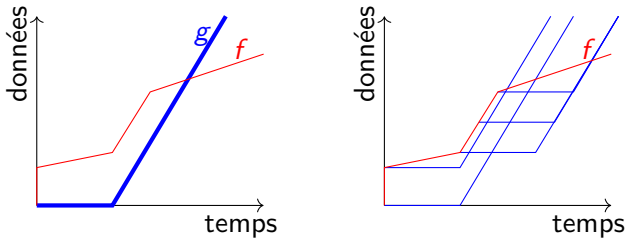
On glisse g sur f et on prend l'enveloppe minimale



Calcul de l'opérateur $*$

Convolution (min,plus) $f * g(t) = \inf_{0 \leq s \leq t} f(s) + g(t-s)$

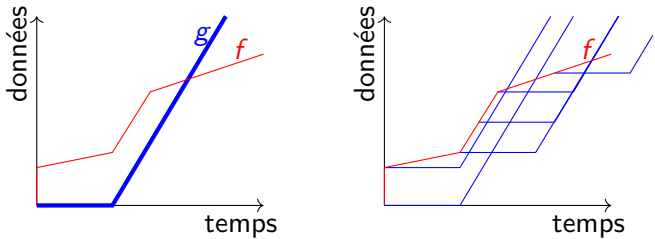
On glisse g sur f et on prend l'enveloppe minimale



Calcul de l'opérateur $*$

Convolution (min,plus) $f * g(t) = \inf_{0 \leq s \leq t} f(s) + g(t-s)$

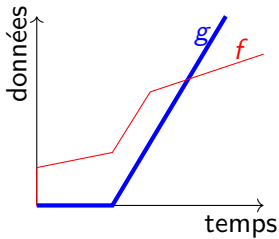
On glisse g sur f et on prend l'enveloppe minimale

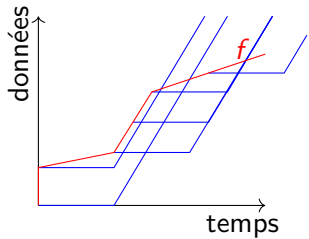


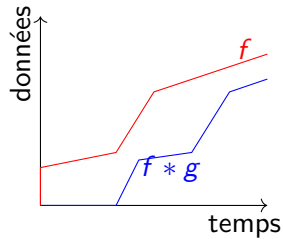
Calcul de l'opérateur $*$

Convolution (min,plus) $f * g(t) = \inf_{0 \leq s \leq t} f(s) + g(t-s)$

On glisse g sur f et on prend l'enveloppe minimale







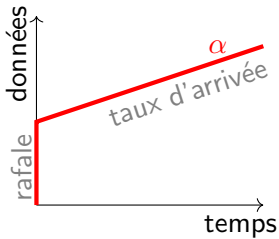
Courbes d'arrivée et courbe de service



Courbe d'arrivée

A est contraint par la fonction α si $\forall 0 \leq s \leq t$,

$$A(t) - A(s) \leq \alpha(t - s).$$



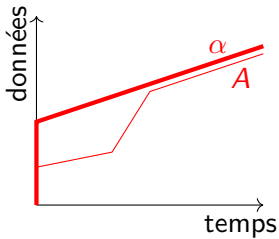
Courbes d'arrivée et courbe de service



Courbe d'arrivée

A est contraint par la fonction α si $\forall 0 \leq s \leq t$,

$$A(t) - A(s) \leq \alpha(t - s).$$



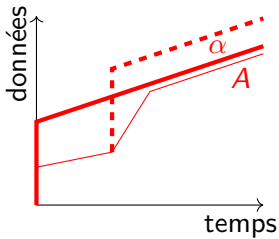
Courbes d'arrivée et courbe de service



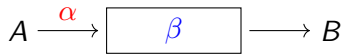
Courbe d'arrivée

A est contraint par la fonction α si $\forall 0 \leq s \leq t$,

$$A(t) - A(s) \leq \alpha(t - s).$$



Courbes d'arrivée et courbe de service



Courbe d'arrivée

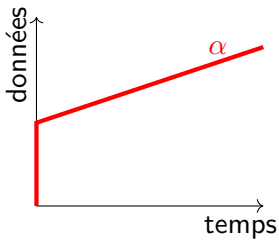
A est contraint par la fonction α si $\forall 0 \leq s \leq t$,

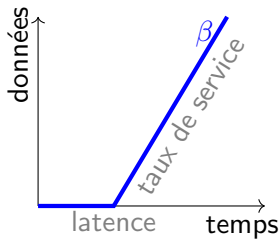
$$A(t) - A(s) \leq \alpha(t-s).$$

Courbe de service

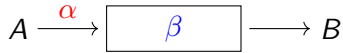
A serveur garantit β pour A si B satisfait

$$B(t) \geq \inf_{0 \leq s \leq t} A(s) + \beta(t-s) = \beta * A(t).$$





Courbes d'arrivée et courbe de service



Courbe d'arrivée

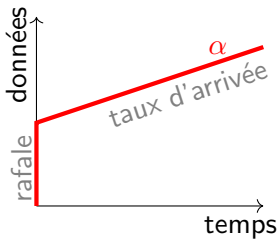
A est contraint par la fonction α si $\forall 0 \leq s \leq t$,

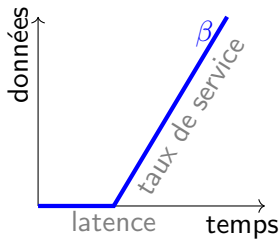
$$A(t) - A(s) \leq \alpha(t-s).$$

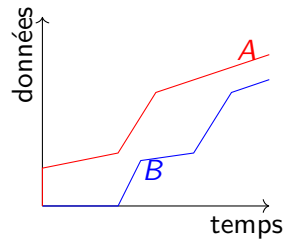
Courbe de service

A serveur garantit β pour A si B satisfait

$$B(t) \geq \inf_{0 \leq s \leq t} A(s) + \beta(t-s) = \beta * A(t).$$







Lien avec l'analyse temps réel

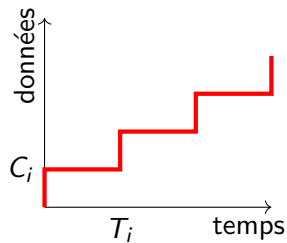
Système de tâches avec priorités fixes préemptives

- n tâches périodiques, si $i < j$, alors i plus prioritaire que j
- Tâche i décrite par (T_i, C_i), sa période et son pire temps d'exécution (WCET).

Question : Calculer le pire temps de réponse (WCRT) R_i pour chaque tâche i.

Courbe d'arrivée

Un paquet de taille C_i tous les T_i : $\alpha_i = \left\lceil \frac{t}{T_i} \right\rceil C_i$, souvent abstraite en $\alpha'_i = C_i + \frac{C_i}{T_i} t$.



Lien avec l'analyse temps réel

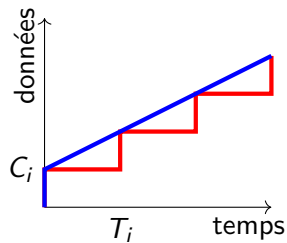
Système de tâches avec priorités fixes préemptives

- n tâches périodiques, si $i < j$, alors i plus prioritaire que j
- Tâche i décrite par (T_i, C_i), sa période et son pire temps d'exécution (WCET).

Question : Calculer le pire temps de réponse (WCRT) R_i pour chaque tâche i.

Courbe d'arrivée

Un paquet de taille C_i tous les T_i : $\alpha_i = \left\lceil \frac{t}{T_i} \right\rceil C_i$, souvent abstraite en $\alpha'_i = C_i + \frac{C_i}{T_i} t$.



Lien avec l'analyse temps réel

Système de tâches avec priorités fixes préemptives

- n tâches périodiques, si $i < j$, alors i plus prioritaire que j
- Tâche i décrite par (T_i, C_i), sa période et son pire temps d'exécution (WCET).

Question : Calculer le pire temps de réponse (WCRT) R_i pour chaque tâche i.

Courbe d'arrivée

Un paquet de taille C_i tous les T_i : $\alpha_i = \left\lceil \frac{t}{T_i} \right\rceil C_i$, souvent abstraite en $\alpha'_i = C_i + \frac{C_i}{T_i} t$.

Courbe de service

Le service peut directement être lié au WCET : $\beta(t) = t$

Lien avec l'analyse temps réel

Système de tâches avec priorités fixes préemptives

- n tâches périodiques, si $i < j$, alors i plus prioritaire que j
- Tâche i décrite par (T_i, C_i), sa période et son pire temps d'exécution (WCET).

Question : Calculer le pire temps de réponse (WCRT) R_i pour chaque tâche i.

Courbe d'arrivée

Un paquet de taille C_i tous les T_i : $\alpha_i = \left\lceil \frac{t}{T_i} \right\rceil C_i$, souvent abstraite en $\alpha'_i = C_i + \frac{C_i}{T_i} t$.

Courbe de service

Le service peut directement être lié au WCET : $\beta(t) = t$

Remarque : Pas les même unités :

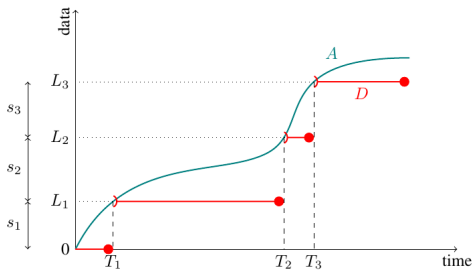
- **NC :** $\beta(t)$ et $\alpha_i(t)$ sont des mesures de quantité de données.
- **RTA :** $\beta(t)$ et $\alpha_i(t)$ sont des mesures de temps
- instance de la tâche $\Leftrightarrow$ paquet

Paquétisation

En pratique, les données sont formées de paquets non-sécables, alors que ce modèle semble plutôt adapté à une modélisation fluide.
On peut se donner la longueur minimale et/ou maximale d'un paquet.

Lemma

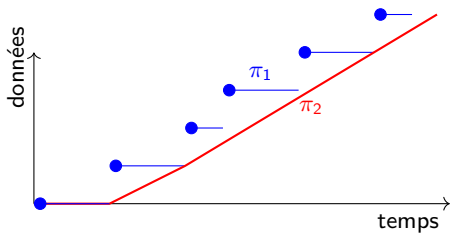
Si β est une courbe de service fluide *d'un serveur, alors $\beta - \ell_{\mathsf{max}}$ est une courbe de service* paquétisée *pour ce serveur.*

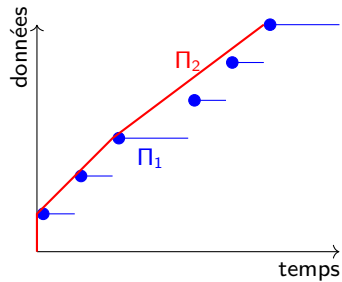


Paquétisation

En pratique, les données sont formées de paquets non-sécables, alors que ce modèle semble plutôt adapté à une modélisation fluide.

1. Les **courbes de paquet** peuvent aussi servir à modéliser de manière plus fine : $\Pi, \pi : \mathbb{R}_+ \to \mathbb{N}$. $\Pi(x)$ (resp.$\pi(x)$) donne le nombre de paquets maximum (resp. minimum) dans une quantité de données x (arrivant à la suite).
2. $\Pi(x) = \frac{1}{\ell_{\min}} 1 + x$ et $\pi(x) = \frac{1}{\ell_{\max}}(x - \ell_{\max})_+$ sont les courbes de paquet les plus simples.



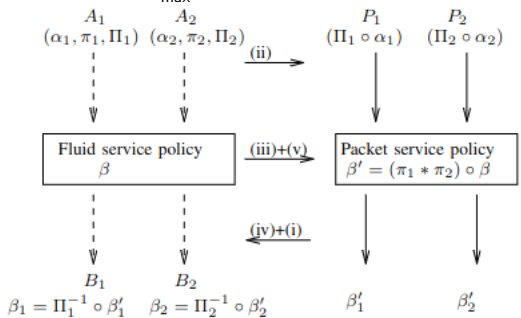


[B, Farhi, Gaujal, Packetization and Aggregate Scheduling, 2011]

Paquétisation

En pratique, les données sont formées de paquets non-sécables, alors que ce modèle semble plutôt adapté à une modélisation fluide.

1. Les **courbes de paquet** peuvent aussi servir à modéliser de manière plus fine : $\Pi, \pi : \mathbb{R}_+ \to \mathbb{N}$. $\Pi(x)$ (resp.$\pi(x)$) donne le nombre de paquets maximum (resp. minimum) dans une quantité de données x (arrivant à la suite).
2. $\Pi(x) = \frac{1}{\ell_{\min}} 1 + x$ et $\pi(x) = \frac{1}{\ell_{\max}}(x - \ell_{\max})_+$ sont les courbes de paquet les plus simples.



[B, Farhi, Gaujal, Packetization and Aggregate Scheduling, 2011]

Politiques d'ordonnancement (ou de service)

Politiques de service fluide

- FIFO (premier arrivé, premier servi)
- Priorités statiques
- GPS: Generalized processor sharing / partage de bande passante
- EDF: Earlier deadline first

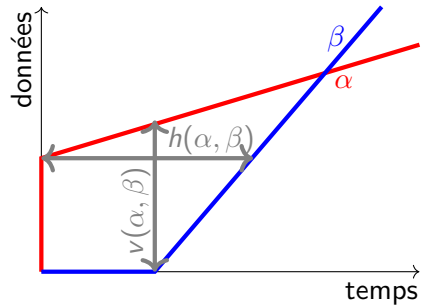
Politiques d'ordonnancement basée sur les paquets

- Les versions paquétisées précédentes...
- RR: Round robin et ses vatriantes (DRR, WRR)
- TSN: time-sensitive networks / CBS - credit-based shapers...
- TDMA: time-division multiple access

Des contraintes aux bornes de performances

Charge maximum :
$B_{\max} = \sup_{t \geq 0} A(t) - B(t)$

Délai maximum:
$D_{\max} = \inf\{d \mid \forall t \in \mathbb{R}_+,\ B(t+d) \geq A(t)\}$



Bornes de performance

- $B_{\max} \leq \alpha \oslash \beta(0) = v(\alpha, \beta) = \sup\{\alpha(t) - \beta(t) \mid t \geq 0\}$
- $D_{\max} \leq h(\alpha, \beta) = \inf\{\forall t \geq 0,\ d \geq 0 \mid \alpha(t) \leq \beta(t+d)\}$ (ordre FIFO).

Résumé

Modèle de flot

- Une courbe d'arrivée
- Un chemin
- Des paquets

Modèle de serveur

- Une courbe d'arrivée (et son type)
- Une politique d'ordonnancement

Contents

Calcul réticulaire : un outil de modèlisation des réseaux

Calcul réticulaire : des outils d'analyse de performance pour les réseaux
- Pour géré un flot traversant plusieurs serveurs
- Pour gérer plusieurs classes de service·
- Analyse dans les réseaux complexes
- Dépendances cycliques

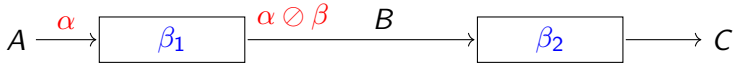
Contents

Calcul réticulaire : un outil de modèlisation des réseaux

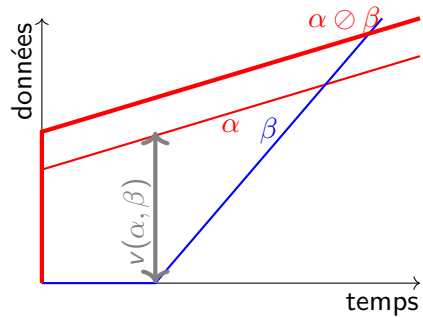
Calcul réticulaire : des outils d'analyse de performance pour les réseaux
- Pour géré un flot traversant plusieurs serveurs
- Pour gérer plusieurs classes de service·
- Analyse dans les réseaux complexes
- Dépendances cycliques

Propagation des courbes d'arrivée

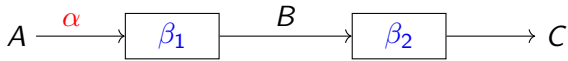
Pour gérer des réseaux (avec plusieurs serveurs), le premier pas est de propager les courbes d'arrivées au serveur suivant.



$\alpha \oslash \beta(t) = \sup_{s \geq 0} \alpha(t+s) - \beta(s)$

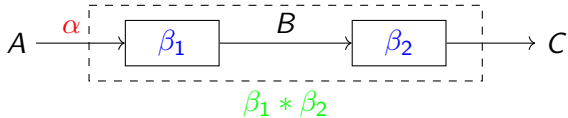


Pay burst only once phenomenon



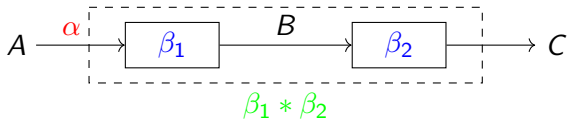
$$C \geq B * \beta_2 \geq (A * \beta_1) * \beta_2$$

Pay burst only once phenomenon



$$C \geq B * \beta_2 \geq (A * \beta_1) * \beta_2 \geq A * (\beta_1 * \beta_2)$$

Pay burst only once phenomenon



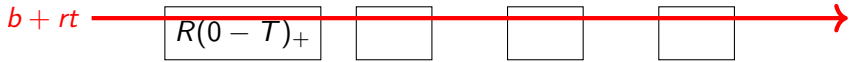
$$C \geq B * \beta_2 \geq (A * \beta_1) * \beta_2 \geq A * (\beta_1 * \beta_2)$$

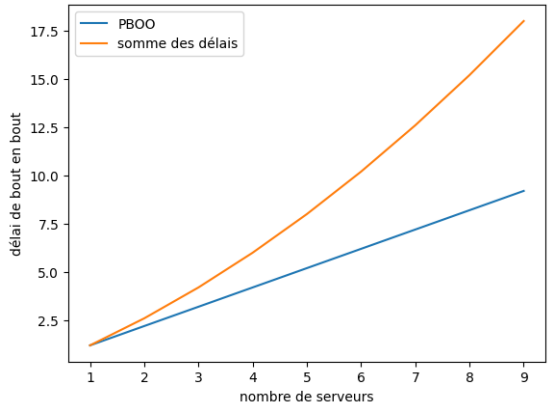
Borne de délai de bout en bout plus précise

$h(\alpha, \beta_1) + h(\alpha \oslash \beta_1, \beta_2) \geq h(\alpha, \beta_1 * \beta_2) \implies d_{1,2} \leq d_1 + d_2$

Le problème est assez simple ici et les calculs algébriques permettent une solution satisfaisante.

PBOO: comparaison





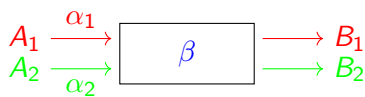
Contents

Calcul réticulaire : un outil de modèlisation des réseaux

Calcul réticulaire : des outils d'analyse de performance pour les réseaux
- Pour géré un flot traversant plusieurs serveurs
- Pour gérer plusieurs classes de service·
- Analyse dans les réseaux complexes
- Dépendances cycliques

Multiplexage arbitraire

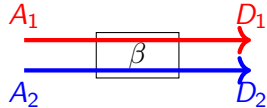
L'étape suivante est de gérer le cas de plusieurs flots traversant le même serveur
Question: trouver une courbe de service individuelle (ou résiduelle) pour chaque flot.



Multiplexage arbitraire

L'ordonnancement le pire pour chaque flot (les bornes de performance seront toujours correctes)

Exemple - service résiduel

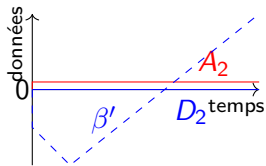


$$D_1 + D_2 \geq (A_1 + A_2) * \beta,$$

donc pour chaque $t \geq 0$, il existe $s \in [0, t]$ tel que

$$\begin{aligned}
(D_1 + D_2)(t) &\geq (A_1 + A_2)(s) + \beta(t - s) \\
D_2(t) &\geq A_2(s) - [D_1(t) - A_1(s)] + \beta(t - s) \\
D_2(t) &\geq A_2(s) - [A_1(t) - A_1(s)] + \beta(t - s) \\
D_2(t) &\geq A_2(s) - \alpha_1(t - s) + \beta(t - s) \\
D_2(t) &\geq A_2(s) - (\beta - \alpha_1)(t - s) \\
D_2(t) &\geq A_2 * (\beta - \alpha_1)(t).
\end{aligned}$$

Exemple - service résiduel



On ne peut pas calculer de courbe résiduelle positive , et donc de délai fini. Il semble donc nécessaire de trouver un modèle plus adapté...

Une difficulté du calcul réticulaire

Courbe de service *simple*

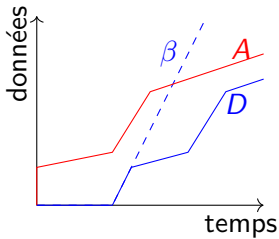
Courbe de service *strict*

Une difficulté du calcul réticulaire

Courbe de service *simple*

$D \geq A * \beta$

$D(t) \geq \inf_{0 \leq s \leq t} A(s) + \beta(t - s)$



- Formulation algébrique
- Simple, modulaire

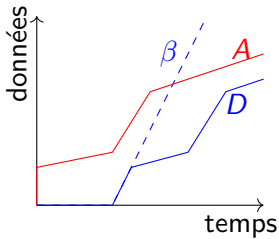
Courbe de service *strict*

Une difficulté du calcul réticulaire

Courbe de service *simple*

$D \geq A * \beta$

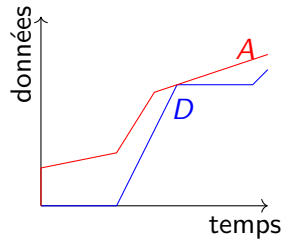
$D(t) \geq \inf_{0 \leq s \leq t} A(s) + \beta(t - s)$



- Formulation algébrique
- Simple, modulaire

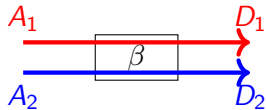
Courbe de service *strict*

Si le système n'est pas vide entre s et t,
$D(t) - D(s) \geq \beta(t - s)$.



- Permet de gérer les flots multiples
- instable pour les opérations

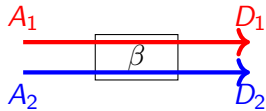
Exemple - service résiduel



s: dernier instant avant t où le système est vide ($A_i(s) = D_i(s)$).

$$
\begin{aligned}
(D_1 + D_2)(t) &\geq (A_1 + A_2)(s) + \beta(t - s) \\
D_2(t) &\geq A_2(s) - [D_1(t) - A_1(s)] + \beta(t - s) \\
D_2(t) &\geq \max(A_2(s), A_2(s) - [A_1(t) - A_1(s)] + \beta(t - s)) \\
D_2(t) &\geq \max(A_2(s), A_2(s) - \alpha_1(t - s) + \beta(t - s) \\
D_2(t) &\geq A_2(s) - (\beta - \alpha_1)_+(t - s) \\
D_2(t) &\geq A_2 * (\beta - \alpha_1)_+(t).
\end{aligned}
$$

Exemple - service résiduel



Theorem (Multiplexage arbitraire)

Si le service est strict, une courbe de service simple pour le flot 2 est

$$(\beta - \alpha_1)_+.$$

$(x)_+ = \max(x, 0)$

Un vrai problème ? Des solutions.

Solution 1 : la pratique

- En pratique, les courbes de service sont strictes (lien à taux fixe)
- Les politiques de service se modélisent bien (cas multi-classe)
- réduction à de réseaux mono-classes FIFO qui ne requièrent pas l'hypothèse de service strict

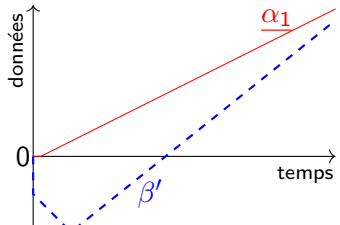
Un vrai problème ? Des solutions.

Solution 1 : la pratique

- En pratique, les courbes de service sont strictes (lien à taux fixe)
- Les politiques de service se modélisent bien (cas multi-classe)
- réduction à de réseaux mono-classes FIFO qui ne requièrent pas l'hypothèse de service strict

Solution 2 : accepter les valeurs négatives

Si on connaît un taux d'arrivée minimum (> 0), alors plus de blocage



On peut faire mieux et calculer de bornes de délai atteignables (fonction de l'arrivée minimum)$\rightarrow$ perte de précision par rapport au service strict (et utilité pratique encore à démontrer)

A. Hamscher, V.-C. Constantin, J. Schmitt, 2024. Extending Network Calculus To Deal With Partially Negative And Decreasing Service Curves.

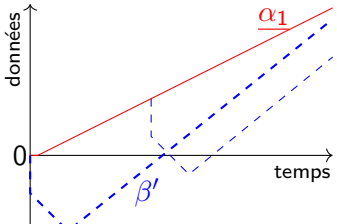
Un vrai problème ? Des solutions.

Solution 1 : la pratique

- En pratique, les courbes de service sont strictes (lien à taux fixe)
- Les politiques de service se modélisent bien (cas multi-classe)
- réduction à de réseaux mono-classes FIFO qui ne requièrent pas l'hypothèse de service strict

Solution 2 : accepter les valeurs négatives

Si on connaît un taux d'arrivée minimum (> 0), alors plus de blocage



On peut faire mieux et calculer de bornes de délai atteignables (fonction de l'arrivée minimum)$\rightarrow$ perte de précision par rapport au service strict (et utilité pratique encore à démontrer)

A. Hamscher, V.-C. Constantin, J. Schmitt, 2024. Extending Network Calculus To Deal With Partially Negative And Decreasing Service Curves.

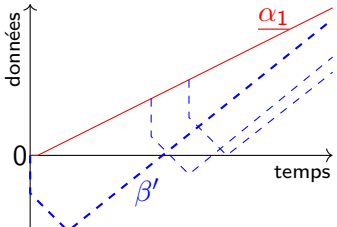
Un vrai problème ? Des solutions.

Solution 1 : la pratique

- En pratique, les courbes de service sont strictes (lien à taux fixe)
- Les politiques de service se modélisent bien (cas multi-classe)
- réduction à de réseaux mono-classes FIFO qui ne requièrent pas l'hypothèse de service strict

Solution 2 : accepter les valeurs négatives

Si on connaît un taux d'arrivée minimum (> 0), alors plus de blocage



On peut faire mieux et calculer de bornes de délai atteignables (fonction de l'arrivée minimum)$\rightarrow$ perte de précision par rapport au service strict (et utilité pratique encore à démontrer)

A. Hamscher, V.-C. Constantin, J. Schmitt, 2024. Extending Network Calculus To Deal With Partially Negative And Decreasing Service Curves.

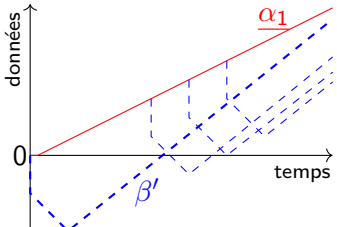
Un vrai problème ? Des solutions.

Solution 1 : la pratique

- En pratique, les courbes de service sont strictes (lien à taux fixe)
- Les politiques de service se modélisent bien (cas multi-classe)
- réduction à de réseaux mono-classes FIFO qui ne requièrent pas l'hypothèse de service strict

Solution 2 : accepter les valeurs négatives

Si on connaît un taux d'arrivée minimum (> 0), alors plus de blocage



On peut faire mieux et calculer de bornes de délai atteignables (fonction de l'arrivée minimum)$\rightarrow$ perte de précision par rapport au service strict (et utilité pratique encore à démontrer)

A. Hamscher, V.-C. Constantin, J. Schmitt, 2024. Extending Network Calculus To Deal With Partially Negative And Decreasing Service Curves.

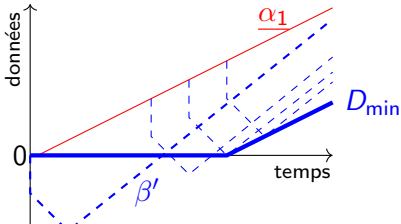
Un vrai problème ? Des solutions.

Solution 1 : la pratique

- En pratique, les courbes de service sont strictes (lien à taux fixe)
- Les politiques de service se modélisent bien (cas multi-classe)
- réduction à de réseaux mono-classes FIFO qui ne requièrent pas l'hypothèse de service strict

Solution 2 : accepter les valeurs négatives

Si on connaît un taux d'arrivée minimum (> 0), alors plus de blocage



On peut faire mieux et calculer de bornes de délai atteignables (fonction de l'arrivée minimum)$\rightarrow$ perte de précision par rapport au service strict (et utilité pratique encore à démontrer)

A. Hamscher, V.-C. Constantin, J. Schmitt, 2024. Extending Network Calculus To Deal With Partially Negative And Decreasing Service Curves.

Priorités statiques

C'est à peu près la même chose

n flots, ordonnés par priorité (comme en RTA), A_i a la courbe d'arrivée α_i

- L'hypothèse service strict est nécessaire.
- **Version fluide :** Le service résiduel ne dépend que des classes plus prioritaires : $\beta_i = (\beta - \sum_{j=1}^{i-1} \alpha_j)_+$.
- **Version paquétisée :** c'est un tout petit peu plus subtil... car on considère en général le cas non-préemptif (un paquet non prioritaire qui a commencé son service le termine quoi qu'il arrive) $\beta_i = (\beta - \sum_{j=1}^{i-1} \alpha_j - \max_{j \geq i} \ell_j)_+$.
- Le service résiduel est strict.

Lien avec l'analyse temps réel

Système de tâches avec priorités fixes préemptives

- n tâches périodiques, si $i < j$, alors i plus prioritaire que j
- Tâche i décrite par (T_i, C_i), sa période et son pire temps d'exécution (WCET).

Question : Calculer le pire temps de réponse (WCRT) R_i pour chaque tâche i.

- Ici, le pire temps de réponse correspond au délai.
- Le résultat est connu : R_i est le délai d_k de l'instance k est $t - (k-1)T_i$ où t est plus petit point fixe de l'équation

$$t = kC_i + \sum_{j<i} \left\lceil \frac{t}{T_j} \right\rceil C_j, \qquad d = \max_k d_k,$$

à mettre en parallèle avec la distance horizontale entre la courbe d'arrivée de i et le service résiduel pour i.

Système de tâches avec priorités fixes préemptives

Courbe d'arrivée de la tâche i

$\alpha(t) = \left\lceil \frac{t}{T_j} \right\rceil C_j$, que l'on peut approximer linéairement par $\alpha_\ell(t) = C_i + \frac{C_i}{T_i} t$.

Courbe de service du processeur

$\beta(t) = t$

$$t = kC_i + \sum_{j<i} \left\lceil \frac{t}{T_j} \right\rceil C_j, \qquad d = \max_k t - (k-1)T_i,$$

peut se réécrire en

$$\beta(t) = \alpha_i((k-1)T_i+) + \sum_{j<i} \alpha_j(t), \quad \text{ou} \quad \alpha_i((k-1)T_i+) = \beta(t) - \sum_{j<i} \alpha_j(t),$$

et on retrouve la formulation souhaitée (en prenant l'hypothèse que le pire cas est obtenu à un temps kT_i+, qui doit être montré en RTA, et se vérifie facilement en NC).

Deux approches différentes

- RTA s'attache à la précision et l'exactitude des calculs (problème NP-difficile ou complet en général)
- NC s'attache en général à l'efficacité du calcul. on ne peut évidemment pas as changer la classe de complexité, mais utiliser le modèle *linéaire* pour simplifier les calculs. Perte de précision.
- Toutefois, il existe de techniques pour prendre en compte la taille des paquets et gagner en précision: différence du temps de calcul selon la précision des courbes d'arrivée choisie.

Premier arrivé premier servi (FIFO)

C'est une politique centrale: lorsqu'on a plusieurs classes d'ordonnancement, les flots dans la même classe sont servis dans leur ordre d'arrivée.

- Tous les flots ont le même délai maximum (détail important dans la suite).

Theorem

Considérons un serveur FIFO offrant une courbe de service β, traversé par deux flots avec les courbe d'arrivées α_1 et α_2. Pour tout $\theta \geq 0$, β_θ est une courbe de service résiduel pour le flot 1, avec

$$\beta_\theta = [\beta - (\alpha_2 * \delta_\theta)]_+ \oplus \delta_\theta.$$

Une infinité de courbes de service résiduelles, mais le service strict n'est pas requis.

[Le Boudec, Thiran, Network Calculus: A Theory of Deterministic Queuing Systems for the Internet, 2004]

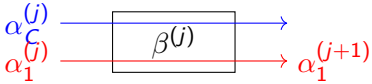
Contents

Calcul réticulaire : un outil de modèlisation des réseaux

Calcul réticulaire : des outils d'analyse de performance pour les réseaux
- Pour géré un flot traversant plusieurs serveurs
- Pour gérer plusieurs classes de service·
- Analyse dans les réseaux complexes
- Dépendances cycliques

Méthode générique et algébrique, ordonnancement arbitraire

1. Dans l'ordre topologique des serveurs, pour chaque flot traversant ce serveur:



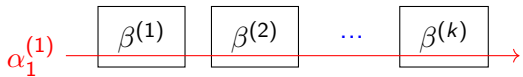
Courbe de service résiduelle:

$$\beta_1^{(j)} = (\beta^{(j)} - \alpha_C^{(j)})_+$$

Courbe d'arrivée en sortie:

$$\alpha_1^{(j+1)} = \alpha_1^{(j)} \oslash \beta_1^{(j)}$$

2. Pour le flot qui nous intéresse :



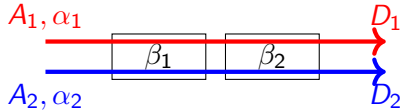
End-to-end service curve:

$$\beta = \beta_1^{(1)} * \beta_1^{(2)} * \cdots * \beta_1^{(k)}$$

3. Délai de bout en bout: $h(\alpha_1, \beta)$,
   Occupation maximale: $v(\alpha_1, \beta)$

- Algorithme efficace : nombre d'opérations linéaire en la taille du réseau (somme des longueurs des flots)
- Bornes de performance très pessimistes
- adaptable à toute politique d'ordonnancement que l'on sait analyser par serveur

La convolution (min,plus) multi-dimensionnelle

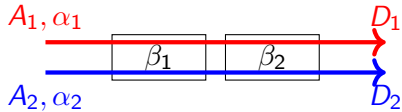


- Une courbe de service *simple :* $\beta_1 * \beta_2$
- Un premier calcul possible :

$$d_1 = h(\alpha_1, [\beta_1 - \alpha_1]_+ * [\beta_2 - (\alpha_2 \oslash (\beta_1 - \alpha_1)_+)]_+)$$

- Mais $(\beta_1 * \beta_2 - \alpha_1)_+$ est une courbe de service pour le flot 2,
- donc on obtient une deuxième borne : $d_2 = h(\alpha_1, [(\beta_1 * \beta_2) - \alpha_2]_+)$
- Dans la plupart des cas, $d_1 \geq d_2$, mais pas toujours.
- pn peut généraliser aux réseaux en tandem. C'est la convolution (min,plus) multi-dimensionnelle aussi appelée *Pay multiplexing only once*:

La convolution (min,plus) multi-dimensionnelle



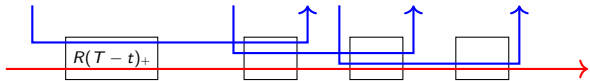
- Une courbe de service *simple :* $\beta_1 * \beta_2$
- Dans la plupart des cas, $d_1 \geq d_2$, mais pas toujours.
- pn peut généraliser aux réseaux en tandem. C'est la convolution (min,plus) multi-dimensionnelle aussi appelée *Pay multiplexing only once*:

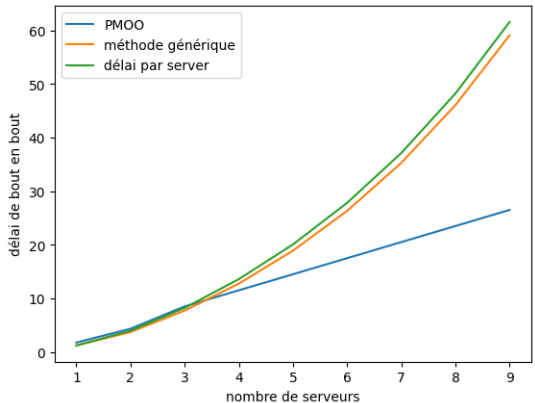


- $\beta_1'(t) = \inf_{u_1+_2+u_3=t} \left(\beta_1(u_1) + \beta_2(u_2) + \beta_3(u_3) - \alpha_2(u_1+u_2) - \alpha_3(u_2+u_3)\right)_+$.

[B., Gaujal, Thierry, Lagrange, 2007]

Comparaison



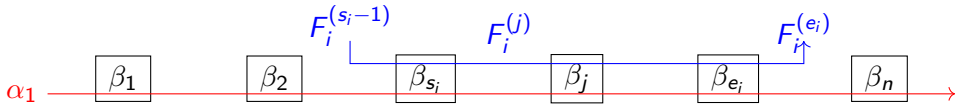


Des méthodes pour calculer les performances pire-cas exactes ?

[Schmitt, Zdarsky, Fidler.
Delay Bounds under Arbitrary Multiplexing:
When Network Calculus Leaves You in the Lurch...2008]

- **Objectif principal :** Réduire le pessimisme qui augmente très fortement avec la taille des réseaux
- **Caveat :** On se place dans des modèles plus restreints (mais similaires au cadre habituel), donc le pessimisme dû à la modélisation peut augmenter.
- **Idée principale :** Garder les processus cumulés autant que possible avant de les borner avec des courbes d'arrivée / service.

Quelques notations

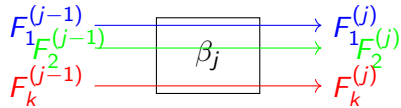


- m flots : le flot i a une courbe d'arrivée α_i linéaire par morceaux et concave (minimum de onctions linéaires;
- Server j offre une courbe de service strict β_j, linéaire par morceaux et convexe (maximum de courbes taux/latence)

Objectifs

- Calculer le délai pire cas de bout en bout du flot 1, qui traverse tout le système ou
- Calculer l'occupation maximale du serveur n.

Des contraintes du calcul réticulaire à la programmation linéaire



$\forall t$, il existe $t_{j-1} = start_j(t)$ (dernier instant avant t où le serveur j est vide) tel que

$$\sum_{i=1}^{k} F_i^{(j)}(t) \geq \sum_{i=1}^{k} F_i^{(j)}(t_{j-1}) + \beta_j(t - t_{j-1})$$

et $F_i^{(j)}(t_{j-1}) = F_i^{(j-1)}(t_{j-1})$.
On choisit maintenant t_n: la date de sortie du bit de données qui satisfait le plus grand délai
$t_{j-1} = start_j(t_j)$.

Le programmal linéaire

Objectif

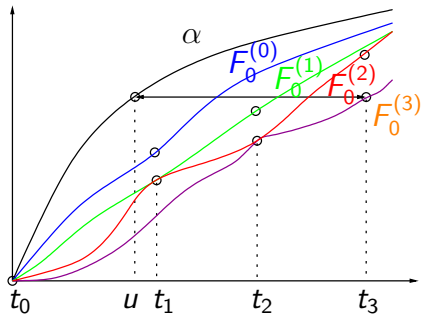
(delay) Maximize : $t_n - u$ with $F_0^{(0)}(u) \leq F_0^{(n)}(t_n)$
(backlog) Maximize : $\sum_i F_i^{(n)}(t_n) - F_i^{(n-1)}(t_n)$

- **Contraintes temporelles** $t_0 \leq t_1 \leq \cdots \leq t_n$; $t_0 \leq u \leq t_n$
- **croissance et causalité des contraintes**
  $F_i^{(j)}(t_\ell) \leq F_i^{(j)}(t_{\ell+1}) \qquad F_i^{(j)}(t_\ell) \geq F_i^{(j+1)}(t_\ell)$
- **Le serveur j est vide à t_j** $F_i^{(j)}(t_{j-1}) = F_i^{(j-1)}(t_{j-1})$
- **Contraintes des courbes de service** $\sum_i F_i^{(j)}(t_j) \geq \sum_i F_i^{(j)}(t_{j-1}) + \beta_j(t_j - t_{j-1})$
- **Contraintes des courbes d'arrivées** $F_i^{(s_i-1)}(t_\ell) - F_i^{(s_i-1)}(t_{\ell-r}) \leq \alpha_i(t_\ell - t_{\ell-r})$;
  $F_0^{(0)}(u) - F_0^{(0)}(t_0) \leq \alpha_0(u - t_0)$

Performances pire-cas dans les réseaux en tandem

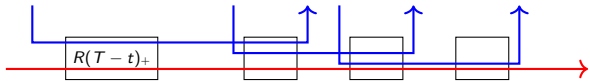
Theorem (B, Jouhet, Thierry, 2010)

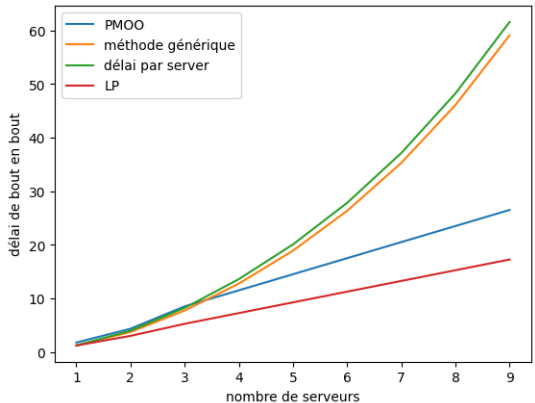
Soit un réseau en tandem avec m flots et n serveurs. Alors, pour un flot i (resp. serveur j), il existe un programme linéaire avec $O(mn)$ variables et $O(mn^2)$ contraintes tel que sa solution optimale de ce programme est le délai pire-cas du flot i (resp. occupation maximale du serveur j)



- On peut généraliser aux réseaux sans dépendance cyclique en introduisant plusieurs programmes linéaires (générer les ordres possibles sur les dates
- Ce problème général est NP-difficile
- Le problème reste polynomial pour des arbres

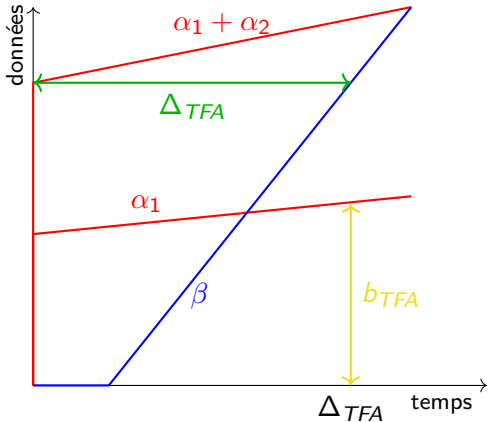
Comparaison

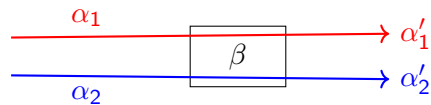




Cas des réseaux FIFO : analyse par agrégation de flot (TFA, Total Flow Analysis)

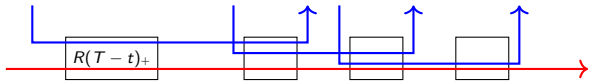
Idea: Dans un serveur, le délai pire-cas d'un flot est le délai pire-cas du flot agrégé.

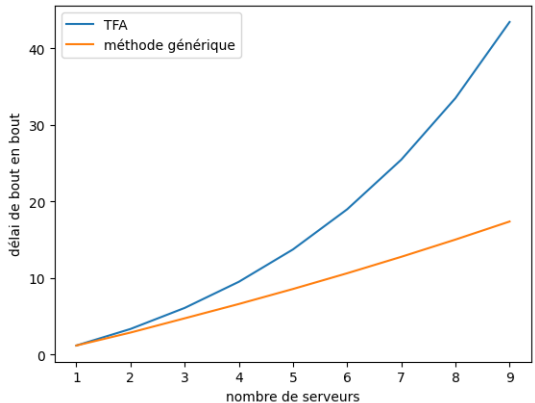




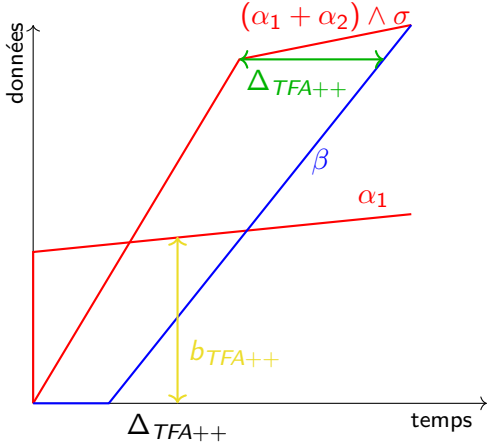
1. Calculer la courbe d'arrivée agrégée $\alpha_1 + \alpha_2$
2. Calculer la borne de délai Δ_{TFA} (distance horizontale maximum entre $\alpha_1 + \alpha_2$ et β
3. $b'_1 = b_{TFA} = b_1 + r_1 \Delta_{TFA}$.

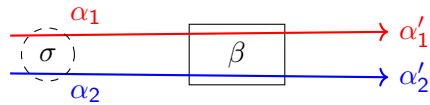
Comparaison





FIFO, amélioration de TFA (TFA++)

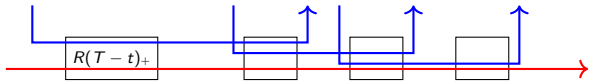


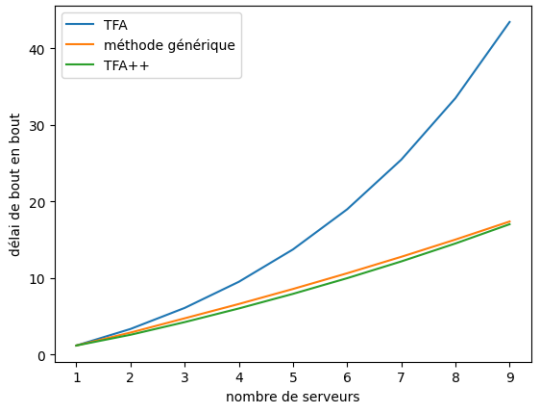


1. Calculer la courbe d'arrivée agrégée $(\alpha_1 + \alpha_2) \wedge \sigma$
2. Calculer la borne de délai Δ_{TFA++} (distance horizontale maximum entre $(\alpha_1 + \alpha_2) \wedge \sigma$ et β
3. $b'_1 = b_{TFA++} = b_1 + r_1 \Delta_{TFA++}$.

[Grieux 2004], [Mifdaoui, Leydier 2017], [Thomas et al. 2019]

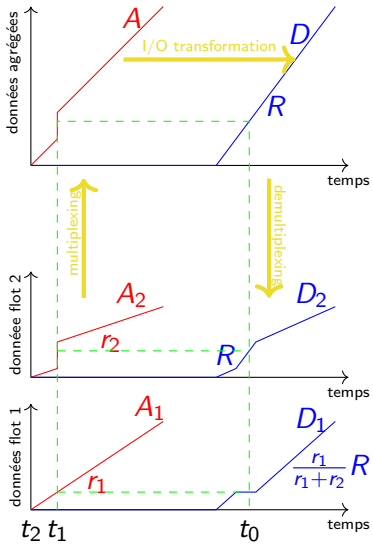
Comparaison





Adaptation de la méthode LP pour les réseaux FIFO

[B., Stea 12]



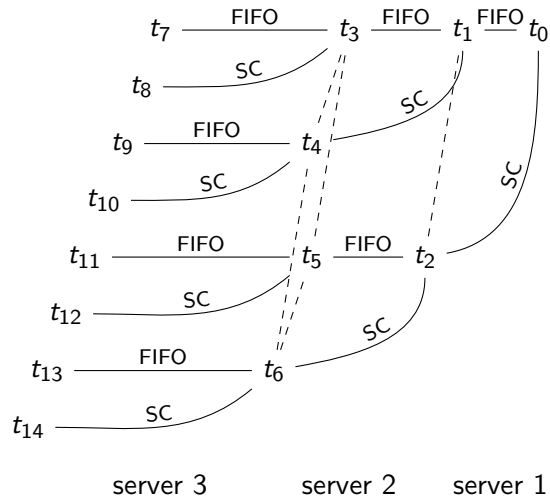
Maximiser $\mathbf{t_0} - \mathbf{t_1}$
Sous les contraintes

- Dates : $\mathbf{t_0} \geq \mathbf{t_1} \geq \mathbf{t_2}$
- Monotonie : $\mathbf{A_i t_0} \geq \mathbf{A_i t_1},\ i \in \{1, 2\}$
- Arrivées : $\mathbf{A_i t_1} - \mathbf{A_i t_2} \leq \alpha_i(\mathbf{t_1} - \mathbf{t_2}),\ i \in \{1, 2\}$
- Service : $\mathbf{D_1 t_0} + \mathbf{D_2 t_0} \geq \mathbf{A_1 t_2} + \mathbf{A_2 t_2} + \beta(\mathbf{t_0} - \mathbf{t_2})$
- FIFO : $\mathbf{D_i t_0} = \mathbf{A_i t_1},\ i \in \{1, 2\}$

- Délais de bout en bout exactes (même hypothèses)
- Avec un nombre exponentiel de variables booléennes

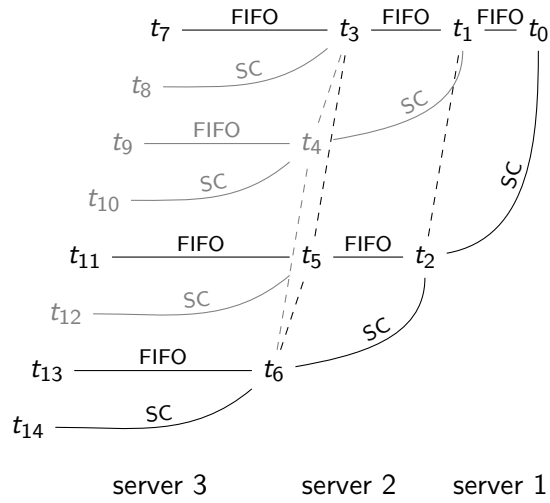
Simplification du programme linéaire

- Nombre de dates exponentiel en la taille du réseau
  - qui doivent être ordonnées, avec des variables booléennes (méthode *big M*)
  - Première approximation/ relaxation: supprimer les variables booléennes, et oublier la monotonie

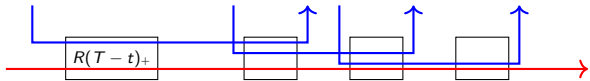


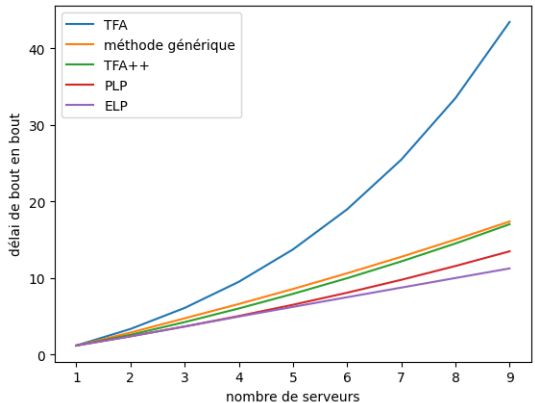
Simplification du programme linéaire

- Nombre de dates exponentiel en la taille du réseau
  - qui doivent être ordonnées, avec des variables booléennes (méthode *big M*)
  - Première approximation/ relaxation: supprimer les variables booléennes, et oublier la monotonie
- Retirer des contraintes de service et garder les contraintes FIFO rend le nombre de dates quadratique
- Remplacer ces contraintes par des délais pré-calculés (TFA++, SFA) pour améliorer les bornes



Comparaison



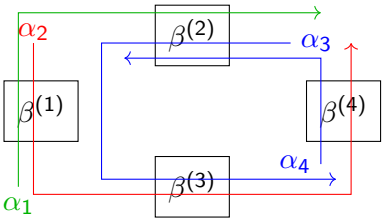


Contents

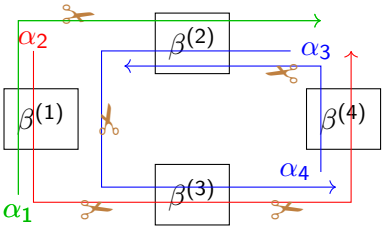
Calcul réticulaire : un outil de modèlisation des réseaux

Calcul réticulaire : des outils d'analyse de performance pour les réseaux
- Pour géré un flot traversant plusieurs serveurs
- Pour gérer plusieurs classes de service·
- Analyse dans les réseaux complexes
- Dépendances cycliques

Réseaux avec des dépendances cycliques



Réseaux avec des dépendances cycliques



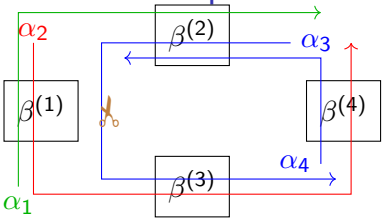
(Les courbes d'arrivées *exogènes* et les courbes de service dont des constantes du problème)

$$\alpha_2^{(3)} = H_2^{(1)}(\alpha_1, \alpha_2)$$
$$\alpha_2^{(4)} = H_2^{(3)}(\alpha_1^{(3)}, \alpha_3^{(3)})...$$

On écrit les équations pour chaque flot en sortie de chaque serveur et on obtient un système d'équations.

$$\boldsymbol{\alpha} = \mathbf{H}(\boldsymbol{\alpha})$$

Réseaux avec des dépendances cycliques



(Les courbes d'arrivées *exogènes* et les courbes de service dont des constantes du problème)

$$\alpha_2^{(3)} = H_2^{(1)}(\alpha_1, \alpha_2)$$
$$\alpha_2^{(4)} = H_2^{(3)}(\alpha_1^{(3)}, \alpha_3^{(3)})...$$

On écrit les équations pour chaque flot en sortie de chaque serveur et on obtient un système d'équations.

Lemme

$$\boldsymbol{\alpha} = \mathbf{H}(\boldsymbol{\alpha})$$

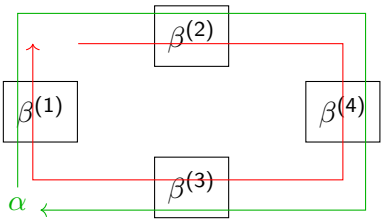
Si le système est stable, alors il existe une famille $\boldsymbol{\alpha} = (\alpha_{i,j})_{i,j}$ de courbes d'arrivées pour les flots $(F_i^{(j)})$ telle que $\boldsymbol{\alpha} \leq \mathbf{H}(\boldsymbol{\alpha})$.

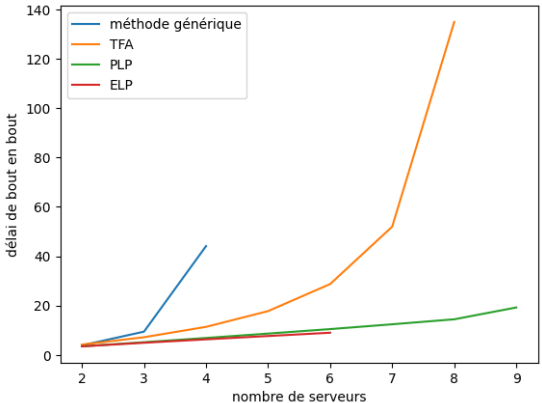
Dans un cadre linéaire, c'est un système linéaire : $\mathbf{b} = M\mathbf{b} + N$.

Programmation linéaire: minimiser les coupes

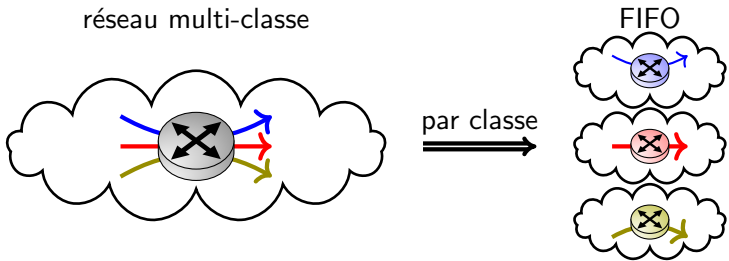
On prouve qu'il existe un unique point fixe, qui donne des courbes d'arrivées et des bornes de délai valides (pas de délai exact)

Comparaison





Analyse de topologies complexes



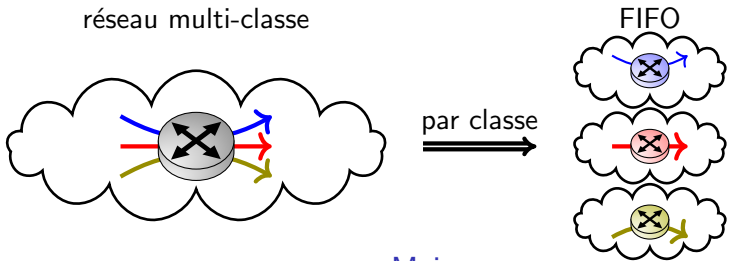
Approche typique

1. Calculer une courbe de service par classe de trafic et par serveur

$\Rightarrow$ des réseaux FIFO indépendants (un par classe)

2. Calcul des bornes de délai par pour chaque réseau FIFO.

Analyse de topologies complexes



Approche typique

1. Calculer une courbe de service par classe de trafic et par serveur

⇒ des réseaux FIFO indépendants (un par classe)

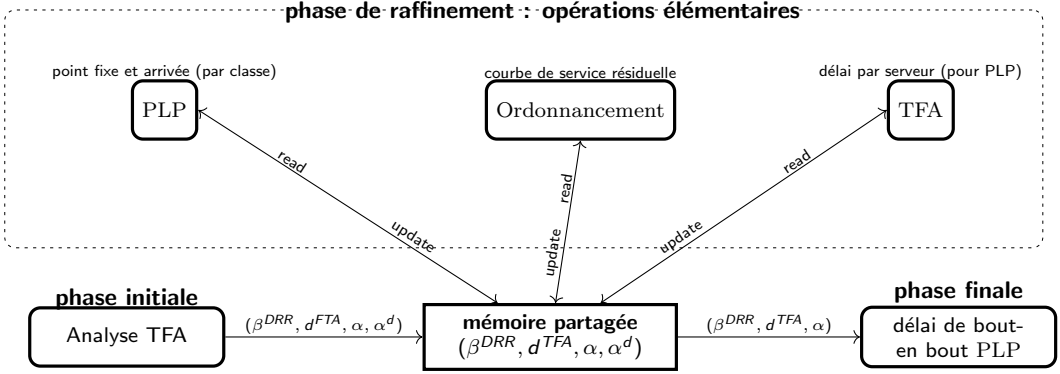
2. Calcul des bornes de délai par pour chaque réseau FIFO.

Mais...

- Les travaux récents montrent qu'utiliser les bornes d'arrivées de toutes les classes dans l'ordonnancement peut améliorer les courbes de service résiduelles

⇒ Interdépendances entre les classes, de nouvelles analyses sont nécessaires.

Un schema iteratif

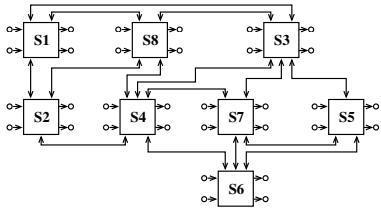


Stratégies pour une convergence rapide

- Alterner une phase PLP, avec la convergence des autres fonctions élémentaires, en parallèle.

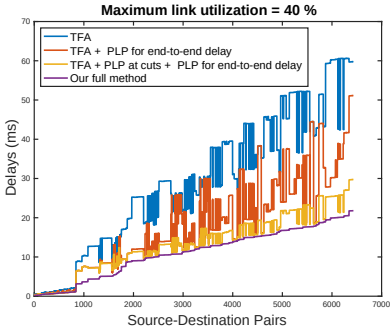
[Tabataee, B. Le Boudec, 2024]

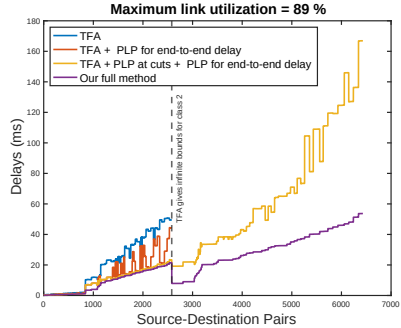
Résultats expérimentaux sur un réseau de taille industrielle



- 96 points d'entrée, 8 routeurs, 984 flots (multicast)
- Taux de service: 1 Gb/s; latence routeur: 16μs.
- Trois classes de flots: (1) critique, (2) multimédia et (3) best-effort.
- politique DRR

classe	1	2	3
flots	128	500	266
quantum	3070	1535	1535





Panco : le calcul réticulaire, le croustillant en plus

Ces méthodes sont (en grande partie) implémentées en Python dans la boîte à outils `Panco`.

- Implémentation en Python `https://github.com/anne-bou/panco`
- Les méthodes basées sur la programmation linéaire (les méthodes TFA et SFA s'implémentent aussi très bien par LP)
- Plusieurs politiques d'ordonnancement : multiplexage arbitraire, FIFO, Priorités statiques, TSN, EDF (DRR et WRR pourraient être bientôt disponibles)
- génère des fichiers LP, et utilisent `lp_solve` et `Cplex` comme solveurs (à installer séparément).

Exemple de fichier : description du modèle

1. Décrire les flots (courbe d'arrivée, chemin)

```
flows = []
flows += [Flow([TokenBucket(1, 2)], [0, 1, 2])]
flows += [Flow([TokenBucket(1, 2)], [0, 1])]
flows += [Flow([TokenBucket(1, 2)], [1, 2])]
```

2. Décrire les serveurs (courbes d'arrivée / shaper)

```
servers = []
servers+=[Server([RateLatency(20,1)],[TokenBucket(0, 30)])]
servers+=[Server([RateLatency(20,1)],[TokenBucket(0, 30)])]
servers+=[Server([RateLatency(20,1)],[TokenBucket(0, 30)])]
```

3. C'est le réseau

```
network = Network(servers, flows)
```

Exemple de fichier : Calcul des bornes

```
print(AdmTfa(network).all_delays)
print(SfaLP(network).all_delays)
print(FifoLP(network, polynomial=True, tfa=True, sfa=True).all_delays)
print(FifoLP(network, polynomial=False).all_delays)
```

Documentation plus précise :
`https://www.di.ens.fr/~bouillar/Panco/html/index.html`

Le calcul réticulaire pas à pas avec PancoSenpai !

Un outil graphique pour

1. dessiner un petit réseau
2. Calculer ses performances
3. Générer le fichier pour une analyse avec Panco

Conclusion

- Calcul réticulaire : un modèle et des outils d'analyse pour calculer des performances pire-cas dans les réseaux.
- Panco : le croustillant en plus !
  `https://github.com/anne-bou/panco`
- PancoSenpai : un pas à pas pour l'utilsation de Panco
  `https://github.com/anne-bou/pancosenpai`