Optimisation d'un algorithme de calcul de bornes de latences pour accélérer la génération de preuves

Florian Delage

La Prépa des INP – Nancy

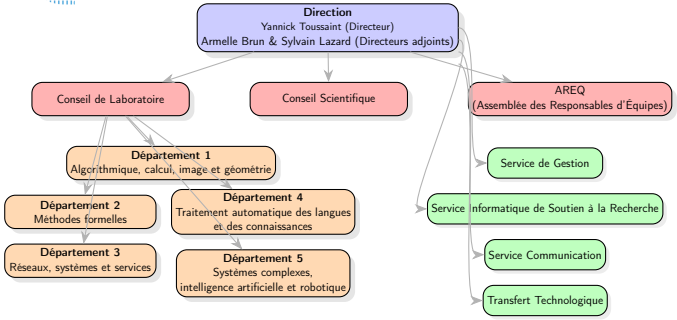
14 juin 2025

Table des Matières

Table des Matières

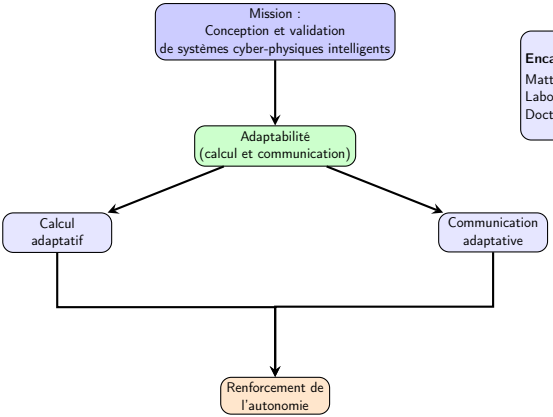
Le Loria

Loria : Laboratoire lorrain de Recherche en Informatique et ses Application



- Créé en 1997
- 28 équipes
- 5 départements
- 500 personnes

Simbiot



Encadrant 1
Matthieu Amet
Laboratoire : Loria (UL)
Doctorant

Encadrant 2
Ludovic Thomas
Laboratoire : Loria (CNRS)
Chargé de Recherche

Table des Matières

Algorithme TFA

Objectif

Calculer les délais maximums (pire cas) que subissent les données dans un réseau à priorités.

Algorithme TFA

Objectif

Calculer les délais maximums (pire cas) que subissent les données dans un réseau à priorités.

Comment ça marche ?

- Les flux ont une priorité (0 à 7) : les plus prioritaires sont traités en premier.
- À chaque nœud du réseau, on calcule combien de temps un flux peut attendre, en prenant en compte :
  - les flux plus prioritaires,
  - le débit disponible,
  - la taille des données.

Zero-Knowledge Proofs (ZKPs)

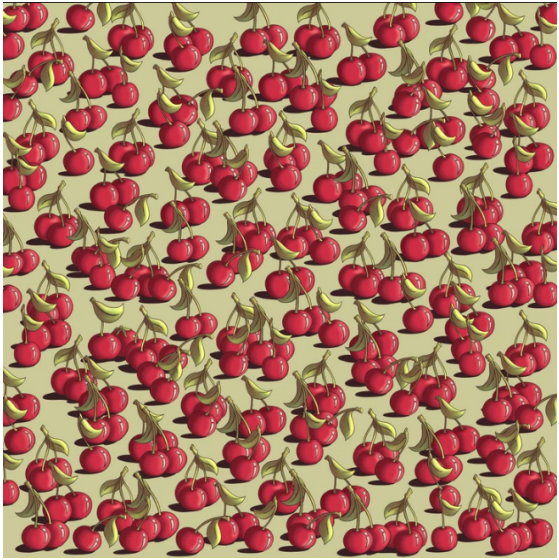


Table des Matières

Conversion des flottants vers des entiers

Entiers

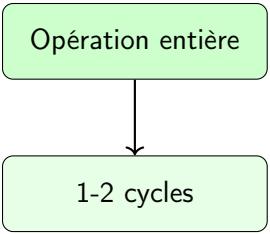
Opération entière

Flottants

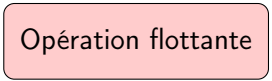
Opération flottante

Conversion des flottants vers des entiers

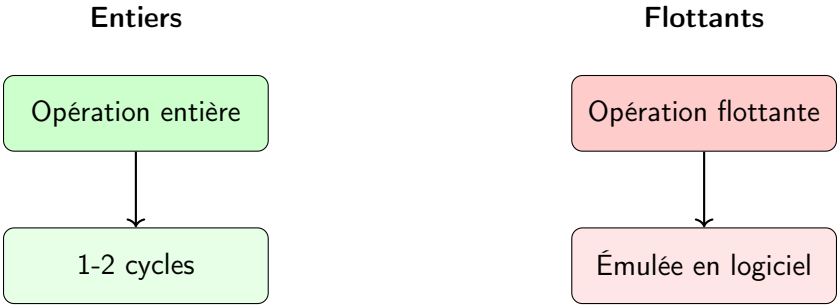
Entiers



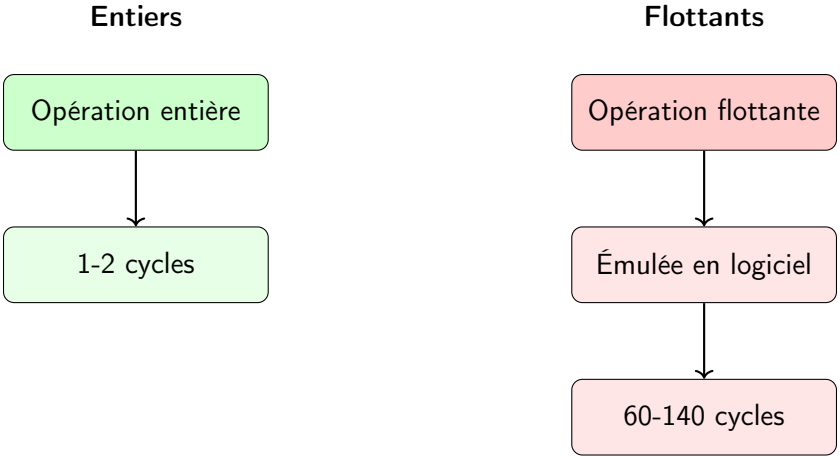
Flottants



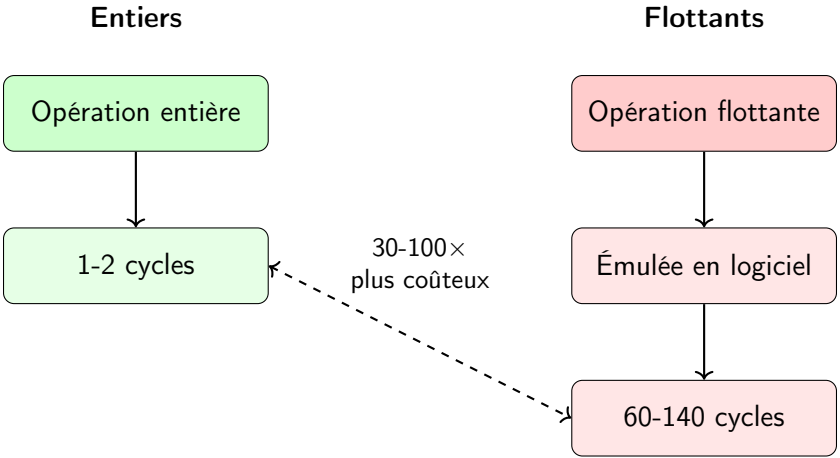
Conversion des flottants vers des entiers



Conversion des flottants vers des entiers

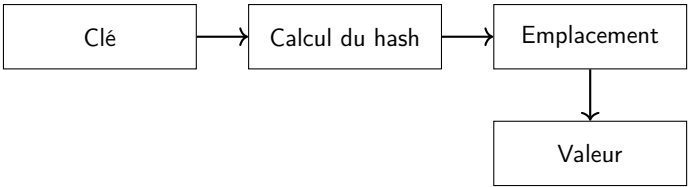


Conversion des flottants vers des entiers



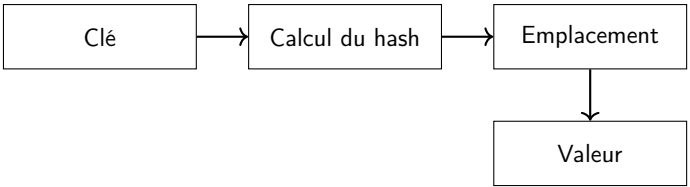
Structures de hachage : HashMap et HashSet

HashMap

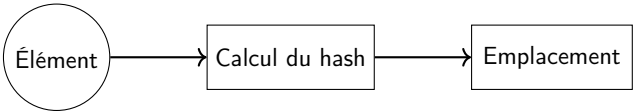


Structures de hachage : HashMap et HashSet

HashMap

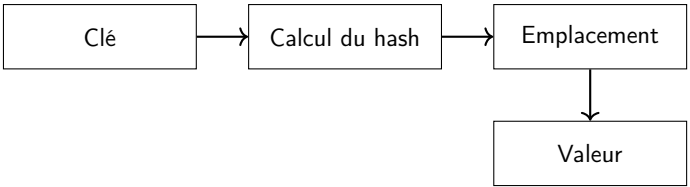


HashSet

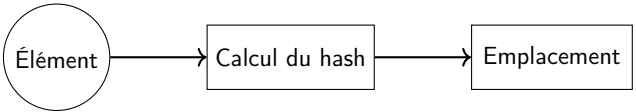


Structures de hachage : HashMap et HashSet

HashMap

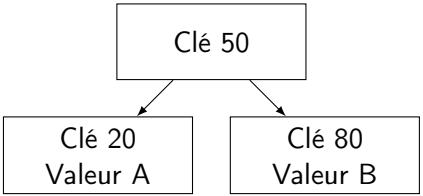


HashSet



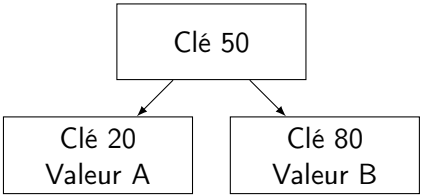
- **HashMap** associe des clés à des valeurs.
- **HashSet** stocke uniquement les valeurs.
- Tous deux utilisent une fonction de hachage.

BTreeMap

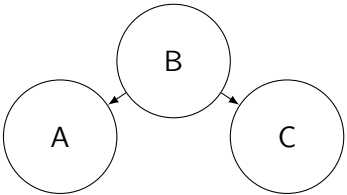


Structures arborescentes : BTreeMap et BTreeSet

BTreeMap

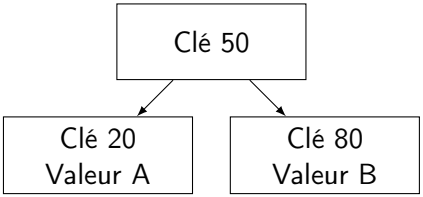


BTreeSet

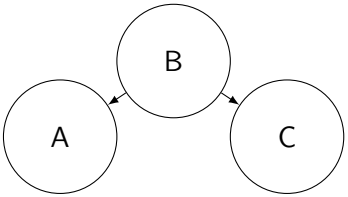


Structures arborescentes : BTreeMap et BTreeSet

BTreeMap



BTreeSet



Résumé :

- **BTreeMap** trie les clés avec leurs valeurs.
- **BTreeSet** trie seulement les valeurs.
- Recherche et insertion par comparaison.

Réduction du nombre de variables

Code non optimisé

Code optimisé

Réduction du nombre de variables

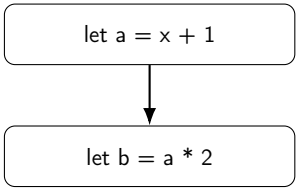
Code non optimisé

```
let a = x + 1
```

Code optimisé

Réduction du nombre de variables

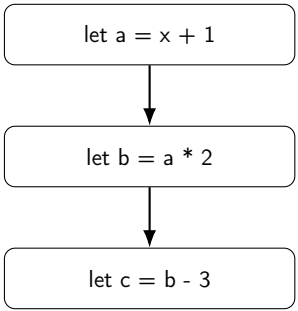
Code non optimisé



Code optimisé

Réduction du nombre de variables

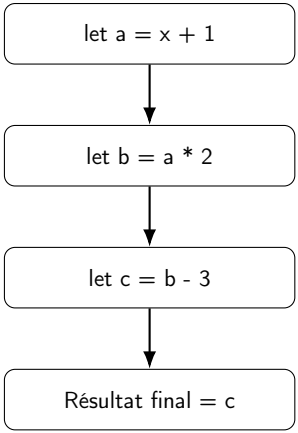
Code non optimisé



Code optimisé

Réduction du nombre de variables

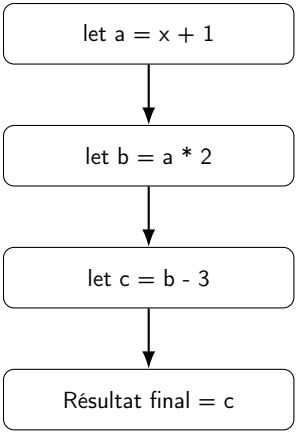
Code non optimisé



Code optimisé

Réduction du nombre de variables

Code non optimisé

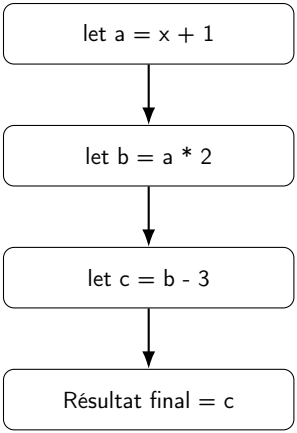


Code optimisé

```
let res = ((x + 1) * 2) - 3
```

Réduction du nombre de variables

Code non optimisé



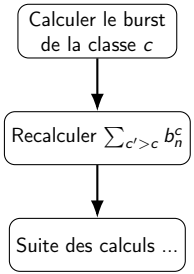
Code optimisé

let res = ((x + 1) * 2) - 3

Résultat final = res

Mémoïsation

Avant



Mémoïsation

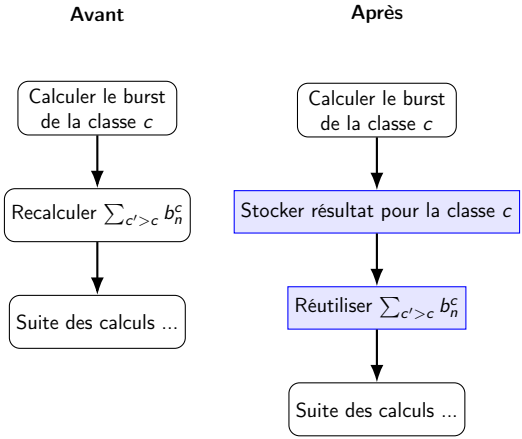
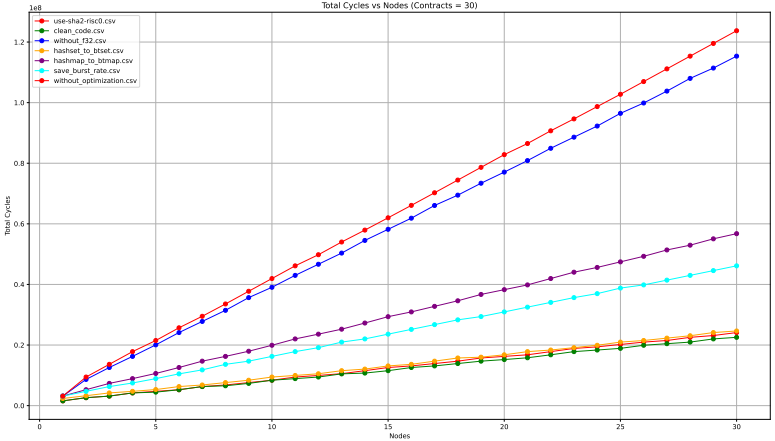


Table des Matières

Graphe de l'évolution du nombre de cycles (nombre de contrats fixe)



Graphe de l'évolution du nombre de cycles (nombre de noeuds fixe)

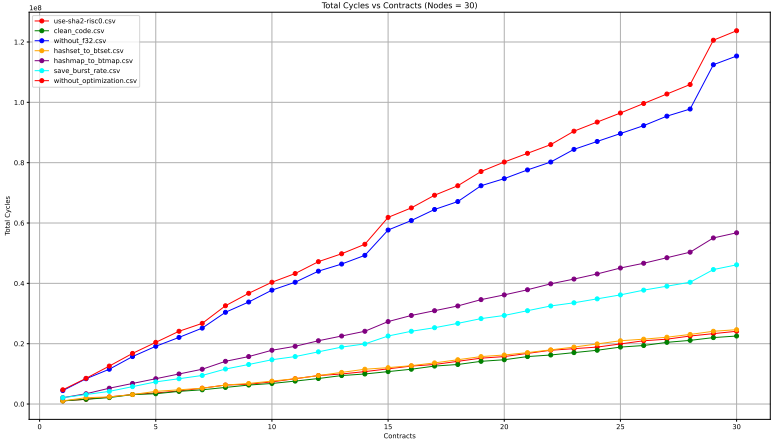


Tableau de synthèse

Optimisation	Cycles moyens	Accélération	Réduction (%)
without_optimization.csv	31941281.56	1.00x	0.00%
without_f32.csv	29764448.71	1.07x	6.82%
hashmap_to_btmap.csv	14783811.13	2.16x	53.72%
save_burst_rate.csv	12123104.14	2.63x	62.05%
hashset_to_btset.csv	6886004.05	4.64x	78.44%
use_sha2_risc0.csv	6505704.11	4.91x	79.63%
clean_code.csv	6158727.40	5.19x	80.72%

Table des Matières

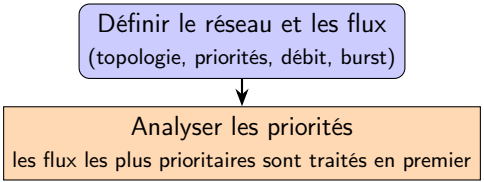
Merci pour votre écoute

Table des Matières

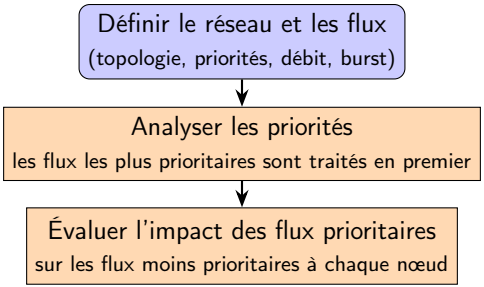
Algorithme TFA (vulgarisé)

Définir le réseau et les flux
(topologie, priorités, débit, burst)

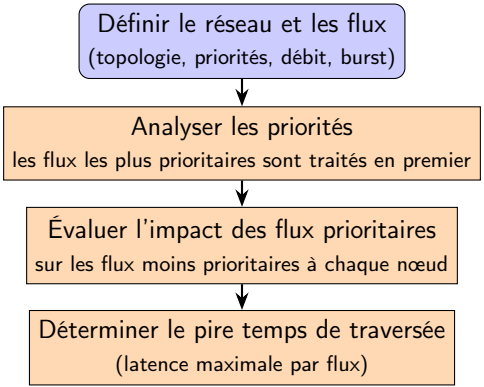
Algorithme TFA (vulgarisé)



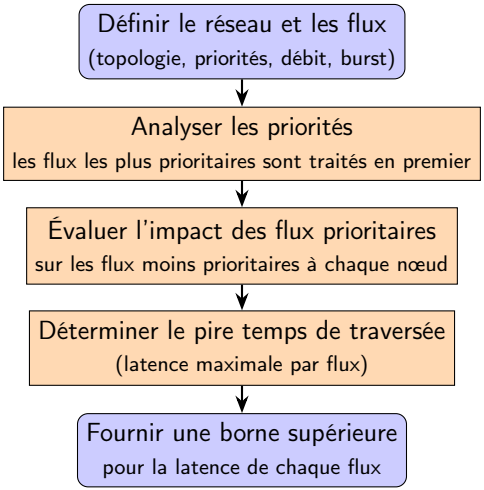
Algorithme TFA (vulgarisé)



Algorithme TFA (vulgarisé)



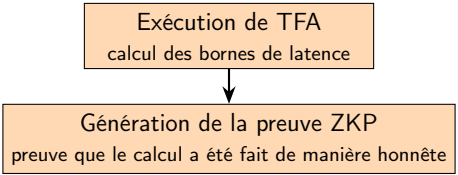
Algorithme TFA (vulgarisé)



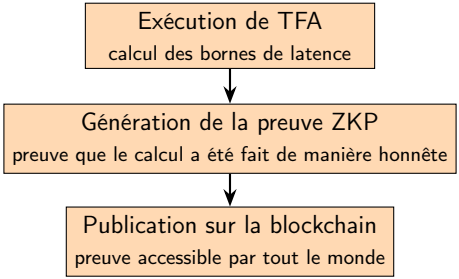
Utilité des preuves

Exécution de TFA
calcul des bornes de latence

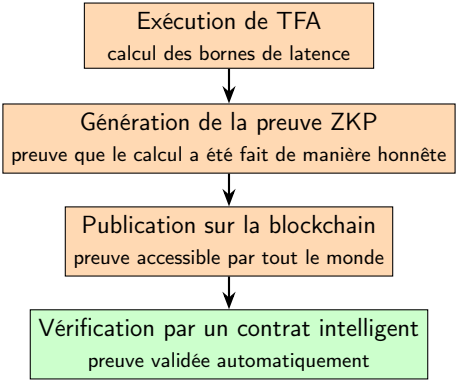
Utilité des preuves



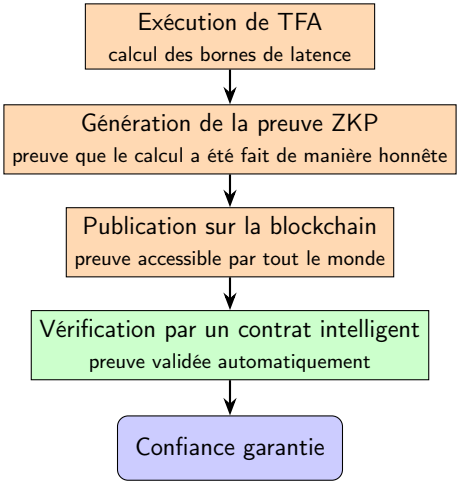
Utilité des preuves



Utilité des preuves



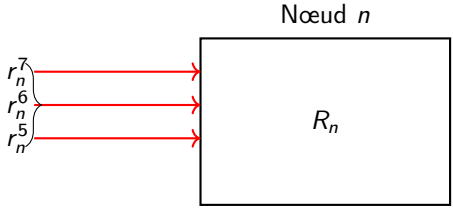
Utilité des preuves



Principe. L'algorithme parcourt les priorités dans l'ordre décroissant, puis chaque nœud du réseau. À chaque étape, il :

Principe. L'algorithme parcourt les priorités dans l'ordre décroissant, puis chaque nœud du réseau. À chaque étape, il :

- Calcule le **débit restant** au nœud.

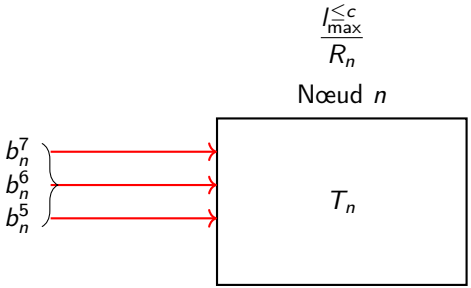


Débit restant pour $c = 4$:

$$R_n^c = R_n - \sum_{c'>c} r_n^{c'}$$

Principe. L'algorithme parcourt les priorités dans l'ordre décroissant, puis chaque nœud du réseau. À chaque étape, il :

- Calcule le **débit restant** au nœud.
- Calcule la **latence de service**.

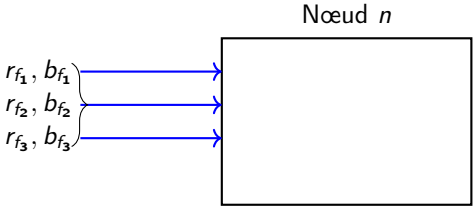


Latence de service pour $c = 4$:

$$T_n^c = T_n + \sum_{c' > c} \frac{b_n^{c'}}{R_n^c} + \frac{I_{\max}^{\leq c}}{R_n}$$

Principe. L'algorithme parcourt les priorités dans l'ordre décroissant, puis chaque nœud du réseau. À chaque étape, il :

- Calcule le **débit restant** au nœud.
- Calcule la **latence de service**.
- Calcule le **débit et le burst**.

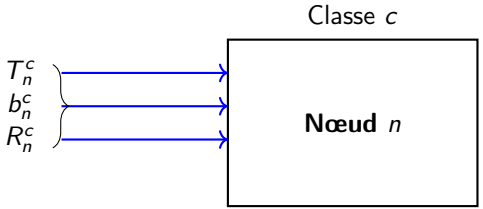


Débit et burst pour c :

$$r_n^c = \sum_{f \in c} r_f \qquad b_n^c = \sum_{f \in c} b_f$$

Principe. L'algorithme parcourt les priorités dans l'ordre décroissant, puis chaque nœud du réseau. À chaque étape, il :

- Calcule le **débit restant** au nœud.
- Calcule la **latence de service**.
- Calcule le **débit et le burst**.
- Calcule la **borne de latence**



Borne de latence pour c :

$$D_n^c = T_n^c + \frac{b_n^c}{R_n^c}$$

Principe. L'algorithme parcourt les priorités dans l'ordre décroissant, puis chaque nœud du réseau. À chaque étape, il :

- Calcule le **débit restant** au nœud.
- Calcule la **latence de service**.
- Calcule le **débit et le burst**.
- Calcule la **borne de latence**
- Propage le **burst** aux nœuds suivants.