

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

Florian Delage

La Prépa des INP – Nancy

12 juin 2025



# Table des Matières

- 1 Présentation du stage
- 2 Compréhension de l'algorithme TFA et des ZKPs (Zero-Knowledge Proofs)
- 3 Travaux effectués
- 4 Évaluation des performances
- 5 Ouverture sur l'algorithme TFA++
- 6 Conclusion

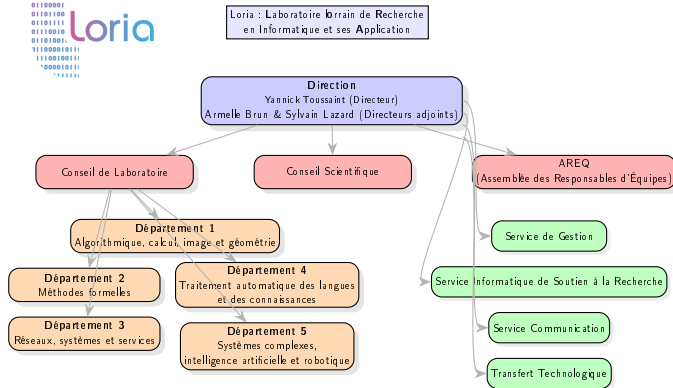


# Table des Matières

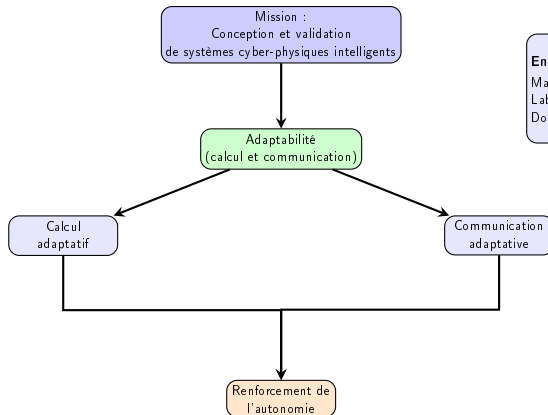
- 1 Présentation du stage
- 2 Compréhension de l'algorithme TFA et des ZKPs (Zero-Knowledge Proofs)
- 3 Travaux effectués
- 4 Évaluation des performances
- 5 Ouverture sur l'algorithme TFA++
- 6 Conclusion







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



## Encadrant 1

Matthieu Amet  
Laboratoire : Loria (UL)  
Doctorant

## Encadrant 2

Ludovic Thomas  
Laboratoire : Loria (CNRS)  
Chargé de Recherche

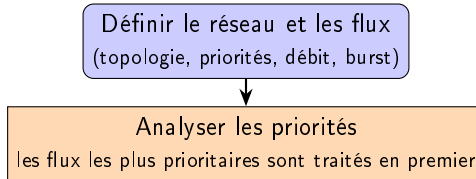
# Table des Matières

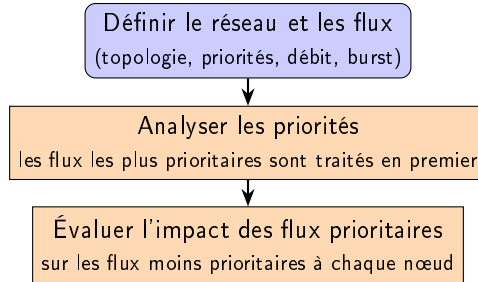
- 1 Présentation du stage
- 2 Compréhension de l'algorithme TFA et des ZKPs (Zero-Knowledge Proofs)
- 3 Travaux effectués
- 4 Évaluation des performances
- 5 Ouverture sur l'algorithme TFA++
- 6 Conclusion

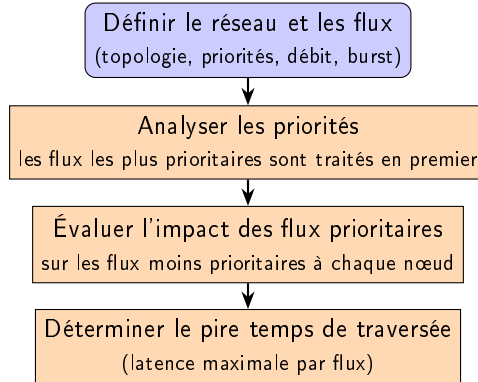


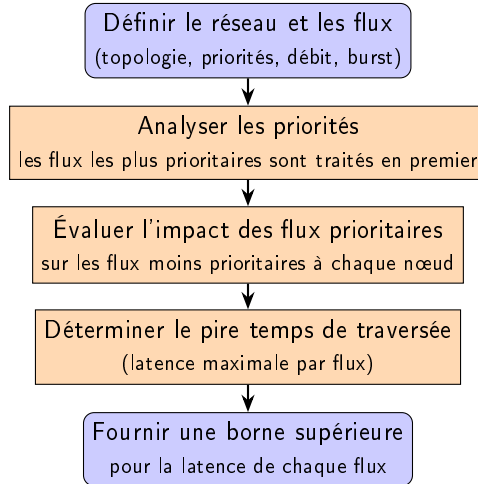
Définir le réseau et les flux  
(topologie, priorités, débit, burst)







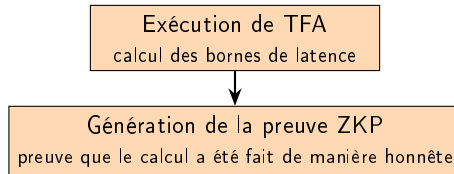


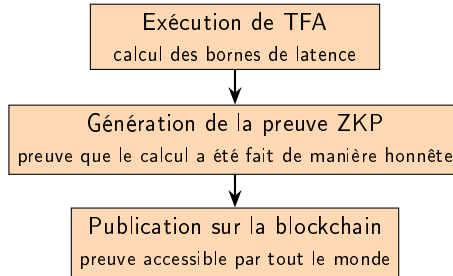


# Zero-Knowledge Proofs (ZKPs)

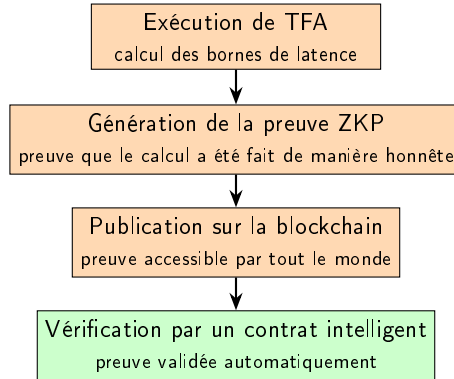


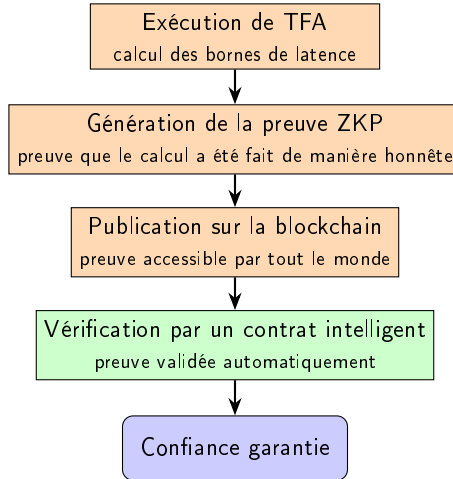
Exécution de TFA  
calcul des bornes de latence











# Table des Matières

- 1 Présentation du stage
- 2 Compréhension de l'algorithme TFA et des ZKPs (Zero-Knowledge Proofs)
- 3 Travaux effectués**
- 4 Évaluation des performances
- 5 Ouverture sur l'algorithme TFA++
- 6 Conclusion



**Entiers**

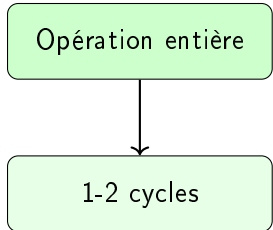
Opération entière

**Flottants**

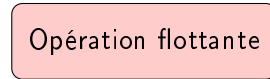
Opération flottante

# Conversion des flottants vers des entiers

## Entiers

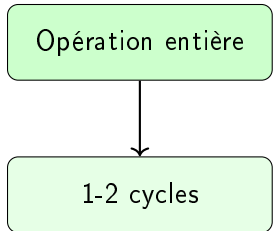


## Flottants

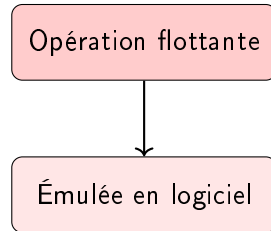


# Conversion des flottants vers des entiers

## Entiers

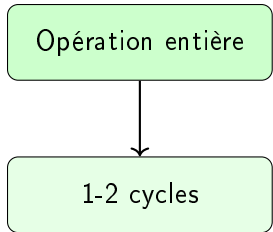


## Flottants

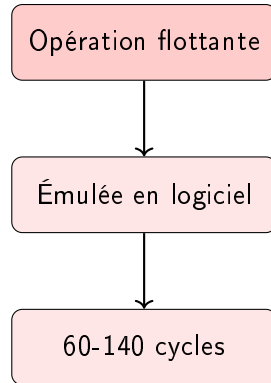


# Conversion des flottants vers des entiers

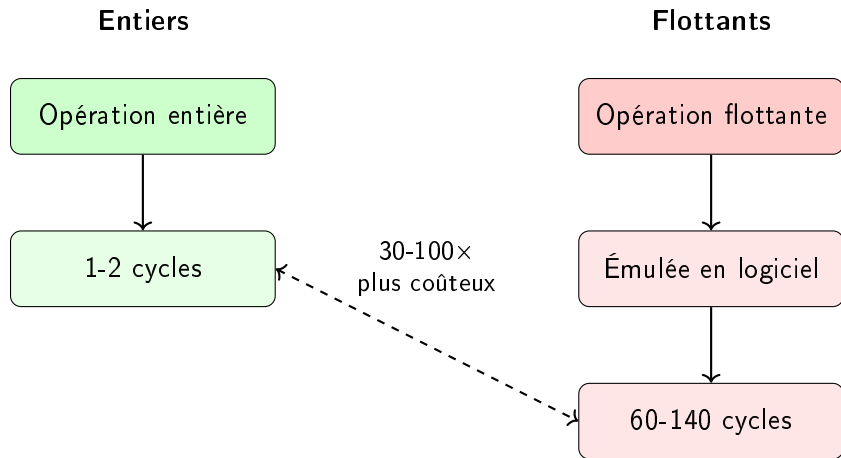
## Entiers



## Flottants



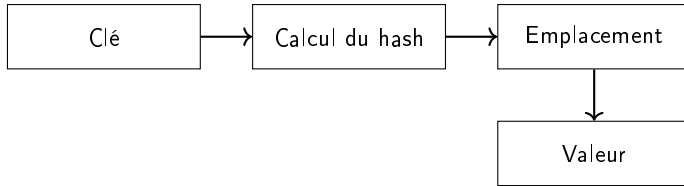
# Conversion des flottants vers des entiers





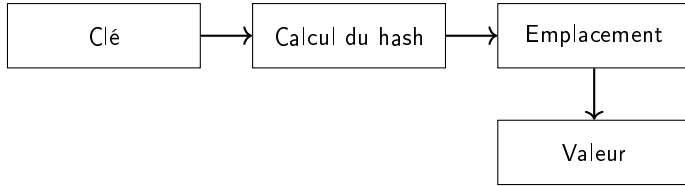
# Structures de hachage : HashMap et HashSet

## HashMap

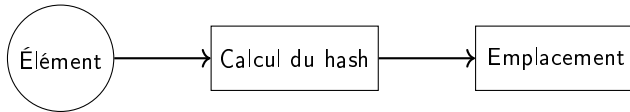


# Structures de hachage : HashMap et HashSet

## HashMap

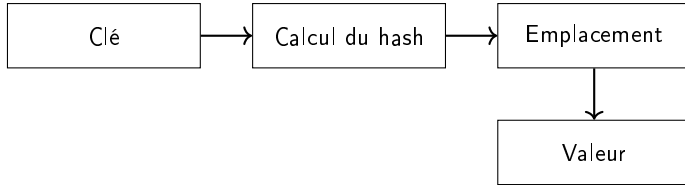


## HashSet

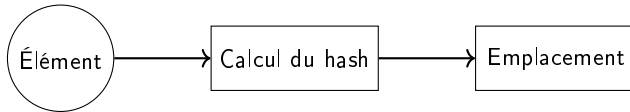


# Structures de hachage : HashMap et HashSet

## HashMap



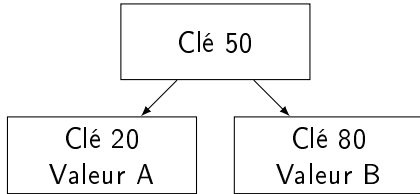
## HashSet



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

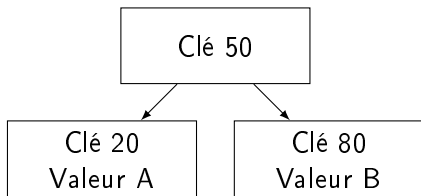
# Structures arborescentes : BTreeMap et BTreeSet

## BTreeMap

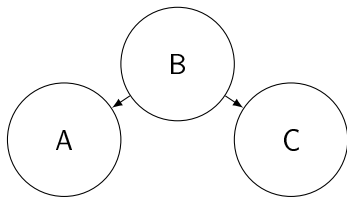


# Structures arborescentes : BTreeMap et BTreeSet

## BTreeMap

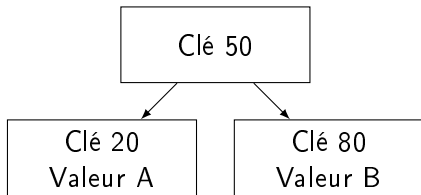


## BTreeSet

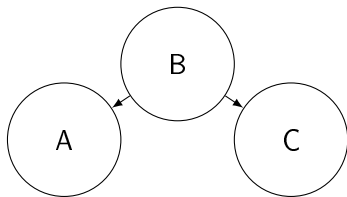


# Structures arborescentes : BTreeMap et BTreeSet

## BTreeMap



## BTreeSet



## Résumé :

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

# Table des Matières

- 1 Présentation du stage
- 2 Compréhension de l'algorithme TFA et des ZKPs (Zero-Knowledge Proofs)
- 3 Travaux effectués
- 4 Évaluation des performances**
- 5 Ouverture sur l'algorithme TFA++
- 6 Conclusion



# Table des Matières

- 1 Présentation du stage
- 2 Compréhension de l'algorithme TFA et des ZKPs (Zero-Knowledge Proofs)
- 3 Travaux effectués
- 4 Évaluation des performances
- 5 Ouverture sur l'algorithme TFA++**
- 6 Conclusion





# Table des Matières

- 1 Présentation du stage
- 2 Compréhension de l'algorithme TFA et des ZKPs (Zero-Knowledge Proofs)
- 3 Travaux effectués
- 4 Évaluation des performances
- 5 Ouverture sur l'algorithme TFA++
- 6 Conclusion**



# Table des Matières

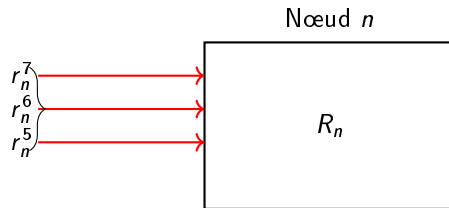
- 1 Présentation du stage
- 2 Compréhension de l'algorithme TFA et des ZKPs (Zero-Knowledge Proofs)
- 3 Travaux effectués
- 4 Évaluation des performances
- 5 Ouverture sur l'algorithme TFA++
- 6 Conclusion



**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.

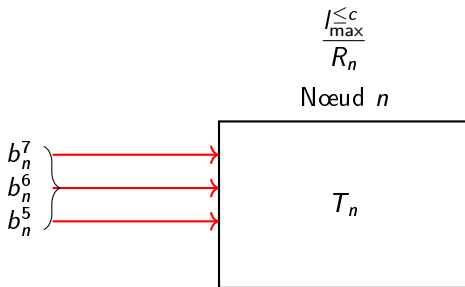


**Débit restant** pour  $c = 4$  :

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

**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**.

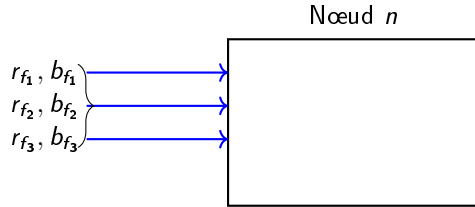


**Latence de service** pour  $c = 4$  :

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

**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**.

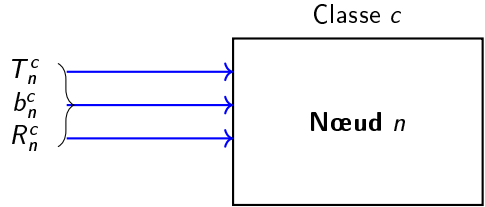


**Débit et burst** pour  $c$  :

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

**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.

