

Optimisation énergétique des cycles Proof-of-Work

Yassir Serroukh

June 8, 2026

- 1 Contexte
- 2 C'est quoi le Green-PoW ?
- 3 Modélisation énergétique du protocole
- 4 Implémentation et simulation
- 5 Résultats et analyses

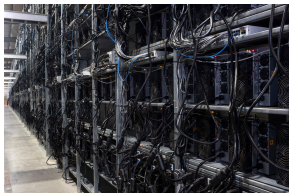


Figure: ferme de minage de cryptomonnaie

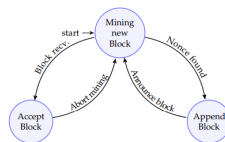


Figure: diagramme d'état d'un mineur dans le PoW original

Defauts du PoW

- Consommation énergétique massive
- Gaspillage de ressources de calcul
- Impact environnemental important

Comment améliorer l'efficacité énergétique du consensus Proof-of-Work tout en garantissant la sécurité et la décentralisation du réseau blockchain ?

Sommaire

- 1 Contexte
- 2 C'est quoi le Green-PoW ?
- 3 Modélisation énergétique du protocole
- 4 Implémentation et simulation
- 5 Résultats et analyses

C'est quoi le Green-PoW ?

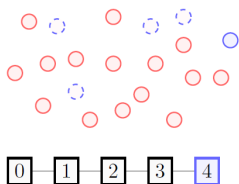


Figure: 1^{er} round de minage (ρ^1)

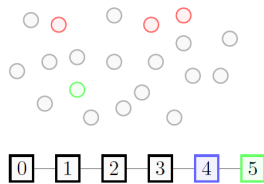


Figure: 2^{eme} round de minage (ρ^2)

Green-PoW vs PoW

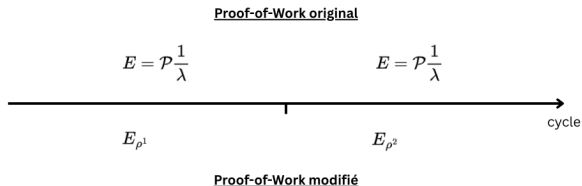


Figure: Consommation énergétique du green-PoW et du PoW au cours de deux epochs

Avantages du green-PoW:

- Moins de calculs inutiles
- Économie d'énergie significative
- Maintien de la sécurité
- Évite certains biais du PoW (ex : fermes de minage)

Modélisation du green-PoW

Temps de retard η

η est un paramètre temporel qui détermine quelle marge de retard est acceptée pour qu'un mineur soit considéré comme "finaliste" après le premier gagnant du round 1.

- Permet de sélectionner un petit groupe de mineurs proches du succès, évitant ainsi que tous recommencent depuis zéro.

Timeout

Le timeout est le temps maximal accordé aux finalistes du round 2 pour réussir à miner un bloc.

- Empêche que la phase ρ^2 dure indéfiniment (si les finalistes échouent à trouver une solution).

Description du protocole

Paramètre	Valeur	Effet sur Green-PoW
η	Petite	Peu de finalistes, faible probabilité de succès du round 2 (ρ^2), échecs fréquents.
	Grande	Beaucoup de finalistes, plus de chances de succès en ρ^2 , mais plus de consommation.
timeout	Court	Si ρ^2 trop lent, timeout atteint avant réussite : bloc abandonné = gaspillage du round 1.
	Long	Meilleur taux de réussite de ρ^2 , mais risque de consommer plus en cas d'échec tardif.

Sommaire

- 1 Contexte
- 2 C'est quoi le Green-PoW ?
- 3 Modélisation énergétique du protocole**
- 4 Implémentation et simulation
- 5 Résultats et analyses

Puissance de hachage

h_i représente la proportion de puissance de calcul que possède le mineur i dans l'ensemble du réseau.

- Si $h_i = 0.1$, alors le mineur i possède 10% de la puissance totale du réseau.
- Plus h_i est élevé $\rightarrow$ plus le mineur a de chances de miner rapidement un bloc.

Taux global de génération de blocs λ

λ est le taux de génération de blocs dans le réseau. En général, on le fixe pour que la blockchain crée un bloc toutes les T_0 secondes :

$$\lambda = \frac{1}{T_0}$$

- Pour Bitcoin : $T_0 = 600s \rightarrow \lambda = \frac{1}{600}s^{-1}$

En PoW, chaque mineur m_i dépense en moyenne $\frac{1}{\lambda}$ et consomme de l'énergie E_i , on note par $\mathcal{P}$ la puissance totale du système:

$$E_i = \frac{1}{\lambda} \cdot h_i \cdot \mathcal{P}$$

Consommation du PoW original

L'énergie totale consommée est: $E = \mathcal{P} \frac{1}{\lambda} \sum_{i=1}^n h_i = \mathcal{P} \frac{1}{\lambda}$

Green-Pow: 1^{er} round

L'énergie du round 1 est composée de deux énergies:

Coût énergétique théorique du round 1

$$E_{\rho^1} = E_{\rho^1,1} + E_{\rho^1,2}$$

Avec:

- $E_{\rho^1,1}$: énergie consommée pour trouver le vainqueur
- $E_{\rho^1,2}$: énergie additionnelle afin de choisir les runners-up

Green-PoW: 1^{er} round, $E_{\rho^1,1}$

Puisqu'avant de passer aux finalistes, le green-PoW se comporte comme le PoW normal:

valeur de $E_{\rho^1,1}$

$$E_{\rho^1,1} = \mathcal{P} \frac{1}{\lambda}$$

Green-PoW: 1^{er} round, $E_{\rho^1,2}$

On note par m_s le vainqueur du 1^{er} round, et $m_{s+1}, \dots, m_{s+k}$ les finalistes choisis.

valeur de $E_{\rho^1,2}$

$$E_{\rho^1,2} = \mathcal{P} \sum_{i=s+1}^{s+k} t_i \cdot (1 - h_{prec,i})$$

Avec $h_{prec,i}$ puissance du mineur gagnant + puissance cumulée des finalistes venus avant

Coût énergétique théorique du round 1

$$E_{\rho^1} = \mathcal{P}\left(\frac{1}{\lambda} + \sum_{i=s+1}^{s+k} t_i \cdot (1 - h_{prec,i})\right)$$

Par analogie au Proof-of-Work original, on a:

Coût énergétique théorique du round 2

$$E_{\rho^2} = \mathcal{P} \frac{1}{\lambda} \sum_{i=s+1}^{s+k} h_i$$

On en déduit:

énergie économisée

$$E_{save} = 2E - (E_{\rho^1} + E_{\rho^2})$$

- 1 Contexte
- 2 C'est quoi le Green-PoW ?
- 3 Modélisation énergétique du protocole
- 4 Implémentation et simulation**
- 5 Résultats et analyses

1^{ère} tentative: modélisation linéaire

On modélise le minage comme un processus linéaire, c'est-à-dire :

- Le temps parcouru (jusqu'au timeout) suit la loi: $t_k = 1 - \frac{k}{N}$ avec N le nombre de mineurs
- Entre chaque $[t_k, t_{k+1}]$, on génère une probabilité uniforme p entre $[0, 1]$, si $p > t_k$, un bloc a été généré, sinon on passe à $[t_{k+1}, t_{k+2}]$
- Plus le temps passe, plus il y'a de chance qu'un bloc soit généré.

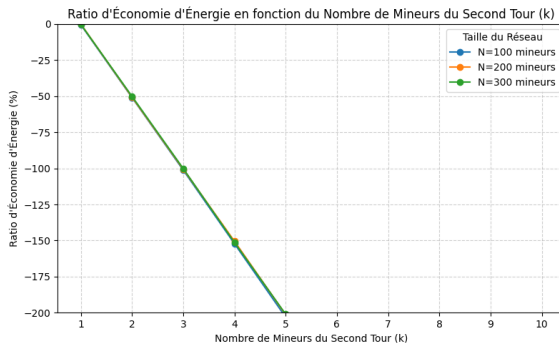


Figure: économie d'énergie VS nombre de mineurs du second round

Conclusion

L'économie d'énergie est négative ! Ça ne marche pas...

2^{ème} tentative: Processus de Poisson

Nous modélisons le minage comme un Processus de Poisson Homogène d'intensité (taux) λ .

- Le nombre d'événements (découverte de blocs) dans un intervalle de temps de longueur t suit une Loi de Poisson de paramètre λt .
- La probabilité de trouver exactement k blocs dans cet intervalle est donnée par la formule de la loi de Poisson :

$$\mathbb{P}[N_t = k] = \frac{(\lambda t)^k e^{-\lambda t}}{k!}$$

Temps d'inter-arrivée

Soit T la variable aléatoire désignant le temps d'inter-arrivée, c'est-à-dire le temps jusqu'à ce que le premier événement (le premier bloc) se produise.

On a alors:

$$\mathbb{P}[T > t] \iff \mathbb{P}[N_t = 0]$$

Et donc la fonction de répartition de T s'écrit :

$$F_T[t] = \mathbb{P}[T \leq t] = 1 - \mathbb{P}[N_t = 0] = 1 - e^{-\lambda t}$$

Simulation des temps d'arrivée

On a alors:

$$p = F_T(t) \iff T = -\frac{1}{\lambda} \log(1 - p)$$

Avec p une probabilité uniforme sur $[0,1]$

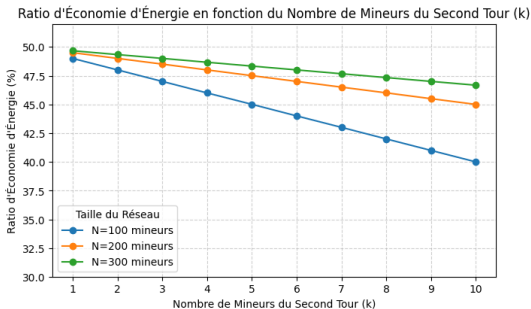


Figure: économie d'énergie VS nombre de mineurs du second round

- Les gains énergétiques sont plus importants dans les petits réseaux.
- Plus il y a de finalistes dans le second round, moins le protocole économise d'énergie.
- L'économie d'énergie se stabilise dans les grands réseaux.

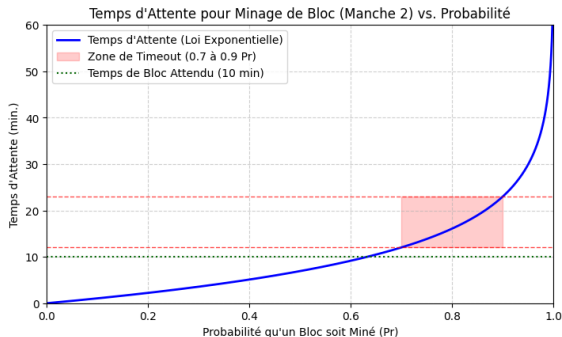


Figure: temps d'attente pour qu'un bloc soit miné VS probabilité de miner un bloc

- Plus la probabilité de miner un bloc est proche de 1, plus le temps explose.
- Le timeout dans Green-PoW permet de couper l'attente à un seuil raisonnable, pour éviter une explosion de la consommation énergétique.
- Le graphique justifie la mise en place d'une limite de temps η après le gagnant du round 1.

Sommaire

- 1 Contexte
- 2 C'est quoi le Green-PoW ?
- 3 Modélisation énergétique du protocole
- 4 Implémentation et simulation
- 5 Résultats et analyses

- Réduction significative de la consommation d'énergie (jusqu'à 50% dans certains cas).
- Structure à deux rounds : seule une fraction des mineurs participe au second round.
- Résultats validés par des simulations statistiques sur plusieurs tailles de réseau.

- Choix critique du nombre de finalistes k : équilibre à trouver entre sécurité et économie.
- Timeout fixe : peut ne pas être optimal selon la situation du réseau.
- Complexité de mise en œuvre : coordination plus poussée entre les mineurs.

Merci pour votre attention

```
import numpy as np
import random
import matplotlib.pyplot as plt
|
# Constantes de Simulation et Paramètres Normalisés
NB_EPOQUES = 100000
TAUX_LAMBDA = 1.0 # Taux de génération de bloc normalisé (1/temps de bloc attendu)
P_TOTAL = 1.0 # Puissance de hachage totale du réseau normalisée
E_BLOC_POW = P_TOTAL / TAUX_LAMBDA # Énergie PoW normalisée par bloc (E = 1)
E_BASE_EPOQUE = 2 * E_BLOC_POW # Énergie PoW normalisée pour deux blocs (2E = 2)

def calculer_ratio_economie_energie(N, k, nb_epochs):
    """
    Simule la consommation d'énergie Green-PoW basée sur le modèle stochastique,
    en supposant une distribution uniforme de la puissance de hachage ( $h_i = 1/N$ ).

    N: Nombre total de mineurs.
    k: Nombre de mineurs du second tour (runners-up).
    nb_epochs: Nombre total d'itérations de la simulation.

    Retourne: Dictionnaire contenant la consommation moyenne et le ratio d'économie.
    """

    # Accumulateur pour l'énergie totale de la 1ère manche
    E_1ere_total = 0.0

    # Calcul de la puissance de hachage individuelle (hypothèse uniforme)
    h_i = 1.0 / N

    # Calcul E_2ème (Moyenne Déterministe, Équation 10):  $E_{2nd} = E * (k * h_i)$ 
    # La somme de la puissance de hachage des runners-up est  $H_k = k/N$ .
    E_2eme_moy = E_BLOC_POW * (k * h_i)
```

```
# Boucle de simulation sur plusieurs époques
for _ in range(nb_epochs):

    # --- Calcul de la 1ère Manche (E_1st) ---
    # E_1st commence avec l'énergie de base pour le premier gagnant (P/lambda = 1)
    E_1ere_epoque = E_BLOC_POW

    # Calcul du surcoût énergétique pour k runners-up
    E_surcout = 0.0

    # h_prec suit la puissance de hachage cumulée du premier gagnant + runners-up précédents.
    # Commence le suivi après la sélection du premier gagnant.
    h_prec = h_i

    for i in range(1, k + 1):
        # i = 1 correspond au premier runner-up.

        # La fraction de puissance de hachage active est H_active = 1 - h_prec
        H_active = 1.0 - h_prec

        if H_active <= 0:
            break

        # Générer la probabilité aléatoire p à partir de Uniform(0, 1)
        p = random.uniform(0.00001, 0.99999)

        # L'énergie de surcoût doit être mise à l'échelle par la puissance d'un seul mineur (h_i = 1/N)
        E_surcout_i = - E_BLOC_POW * (h_i/2) * np.log(1 - p)

        E_surcout += E_surcout_i

    # Mise à jour de h_prec en ajoutant la puissance du runner-up nouvellement sélectionné
    h_prec += h_i
```

```
E_lere_total += (E_lere_epoque + E_surcout)

# Calcul de la consommation énergétique moyenne globale
E_lere_moy = E_lere_total / nb_epochs
E_total_epoque_moy = E_lere_moy + E_2eme_moy

# Calcul du ratio d'économie (Équation 11)
E_economie = E_BASE_EPOQUE - E_total_epoque_moy
ratio_economie = (E_economie / E_BASE_EPOQUE) * 100.0

return {
    'N': N,
    'k': k,
    'E_lere': E_lere_moy,
    'E_2eme': E_2eme_moy,
    'E_Total': E_total_epoque_moy,
    'Ratio_Economie': ratio_economie
}

# Exécution des simulations
tailles_reseau = [100,200,300]
valeurs_k = list(range(1, 11))
resultats_simulation = []

for N in tailles_reseau:
    for k in valeurs_k:
        resultat = calculer_ratio_economie_energie(N, k, NB_EPOQUES)
        resultats_simulation.append(resultat)
```

```
# Traçage des résultats

# Restructurer les données pour le traçage (séparer les résultats par N)
donnees_graphique = {}
for res in resultats_simulation:
    N = res['N']
    if N not in donnees_graphique:
        donnees_graphique[N] = {'k': [], 'ratio_economie': []}

    donnees_graphique[N]['k'].append(res['k'])
    # Accéder à la valeur numérique spécifique 'Ratio_Economie'
    donnees_graphique[N]['ratio_economie'].append(res['Ratio_Economie'])

# Créer le graphique
plt.figure(figsize=(10, 6))

for N, donnees in donnees_graphique.items():
    plt.plot(donnees['k'], donnees['ratio_economie'], marker='o', label=f'N={N} mineurs')

plt.title("Ratio d'Économie d'Énergie en fonction du Nombre de Mineurs du Second Tour (k)")
plt.xlabel('Nombre de Mineurs du Second Tour (k)')
plt.ylabel('Ratio d'Économie d'Énergie (%)')
plt.grid(True, linestyle='--', alpha=0.6)
plt.legend(title='Taille du Réseau')
plt.xticks(valeurs_k)
plt.ylim(30, 50)
plt.show()
```