PRS
Soutenance de projet

Agnès Thouvenin et Estelle Monier
aka LesTryhardeusesDuDimanche

Sommaire

Introduction

- Implanter les mécanismes de TCP sur UDP
- Envoi de fichiers avec 3 scénarios différents
- Objectif : obtenir le meilleur débit possible

Pourquoi le Go ?

- Go routines pour paralléliser simplement
- Gestion automatique de la mémoire contrairement au C
- Performances très légèrement inférieures au C

Pourquoi des paquets de 1500 bytes ?

- MTU d'Ethernet = 1500 bytes

Scénario 1

1°) Explication des principes de notre code

A - Écoute des connexions

Coté serveur

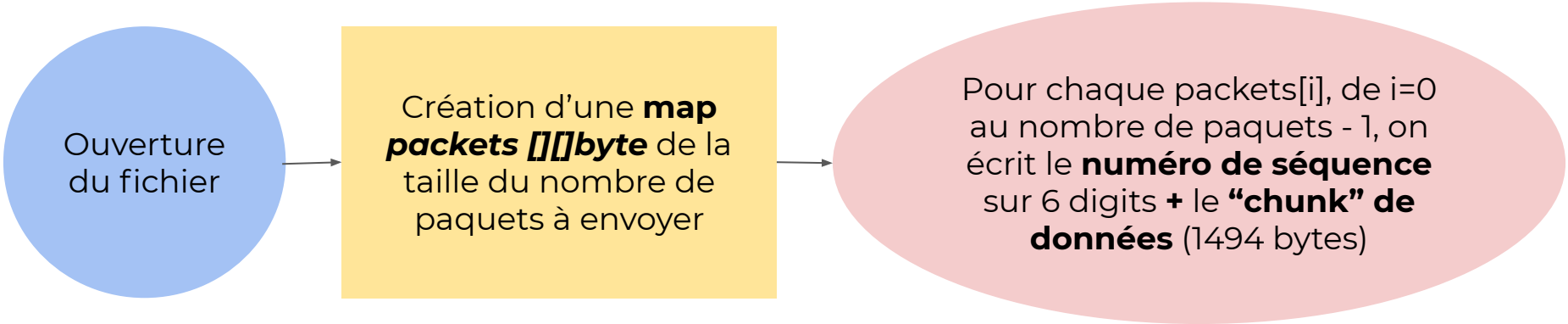
B - Récupération du nom du fichier

Scénario 1

1°) Explication des principes de notre code

C - Envoi du fichier avec la fonction sendFile

1 - Préparation des paquets



Scénario 1

1°) Explication des principes de notre code

C - Envoi du fichier avec la fonction sendFile

2 - Envoi du fichier

Envoi des paquets par *fenêtres*, tant qu'on a pas reçu tous les ACK de la fenêtre, on envoie pas la prochaine. Si *paquet perdu* -> envoi d'une *fenêtre slidée* à partir du paquet perdu.

Fonction send :

- Envoie le paquet en entrée au client (taille adaptée).
- Lance un timer pour ce paquet dans un tableau de timers *timeouts []time.Time*

Fonction window :

- *True* si le paquet courant est dans la fenêtre.
- *False* sinon.
- Calcule les nouvelles bornes de la fenêtre à chaque passage.

Scénario 1

1°) Explication des principes de notre code

C - Envoi du fichier avec la fonction sendFile

2 - Envoi du fichier

Go routine :

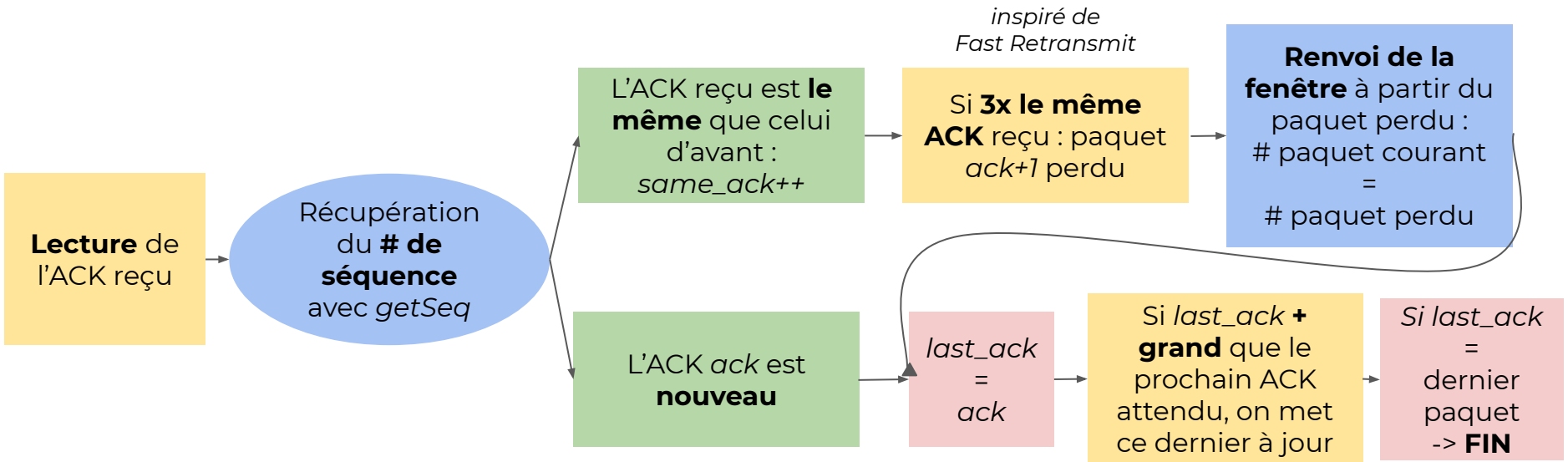
- Boucle toutes les 1 ms
- Si *window* retourne True :
  - envoi du paquet avec *send*
  - incrémentation du # de paquet courant
- Sinon :
  - Si le timeout est atteint pour le paquet dont on attend l'ACK (*next_biggest_ack*) :
    - Paquet perdu, on renvoie une fenêtre depuis ce dernier :
      # de paquet courant = # du paquet perdu

Scénario 1

1°) Explication des principes de notre code

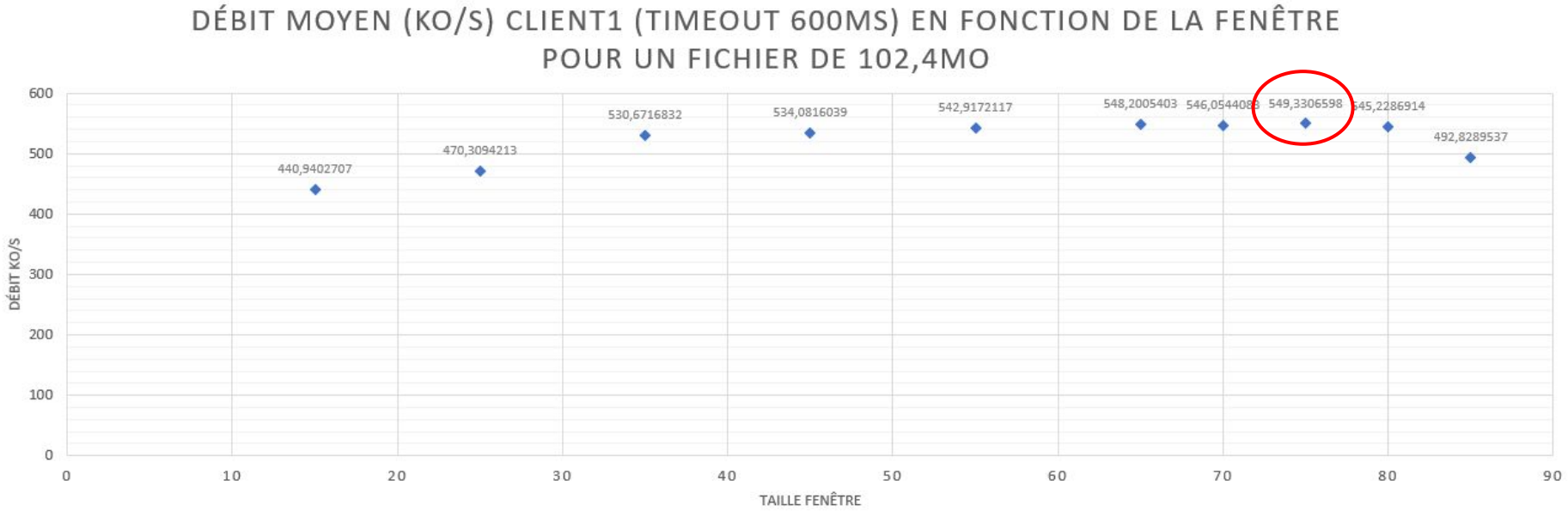
C - Envoi du fichier avec la fonction sendFile

3 - Réception et gestion des ACK (boucle tant qu'on ne les a pas tous reçus)



Scénario 1

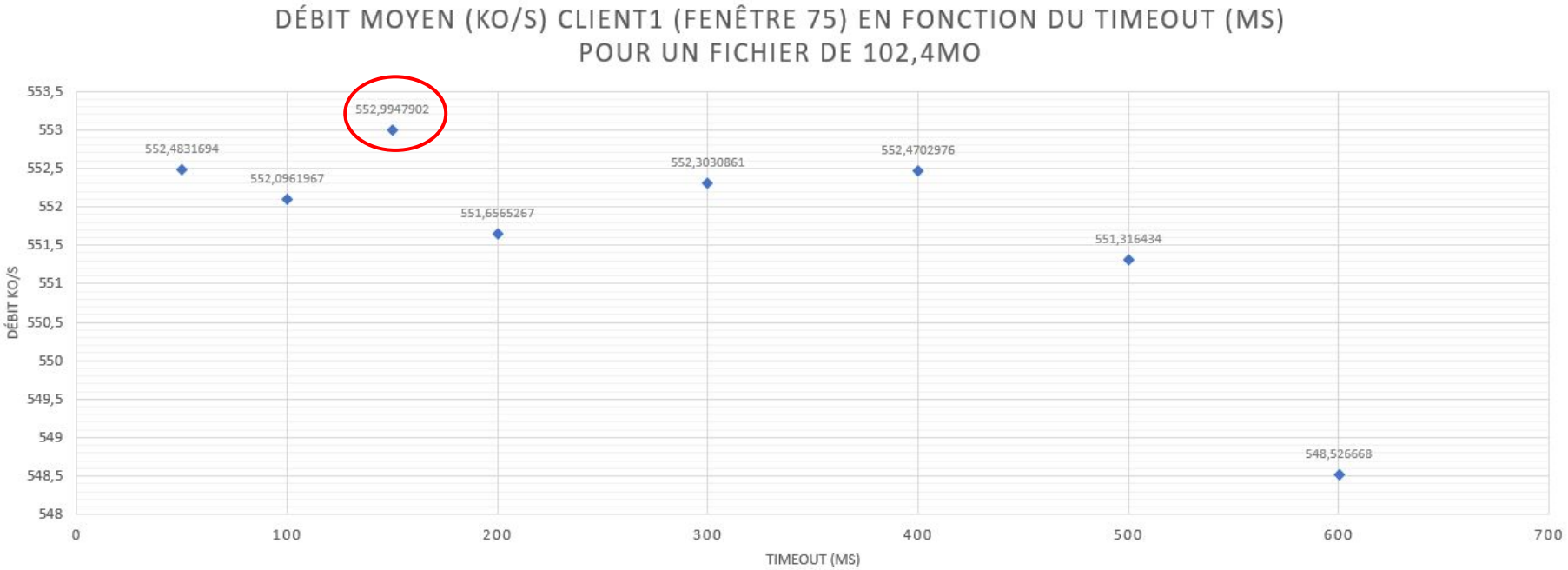
2°) Recherche d'une configuration optimale



Taille de fenêtre optimale : 75

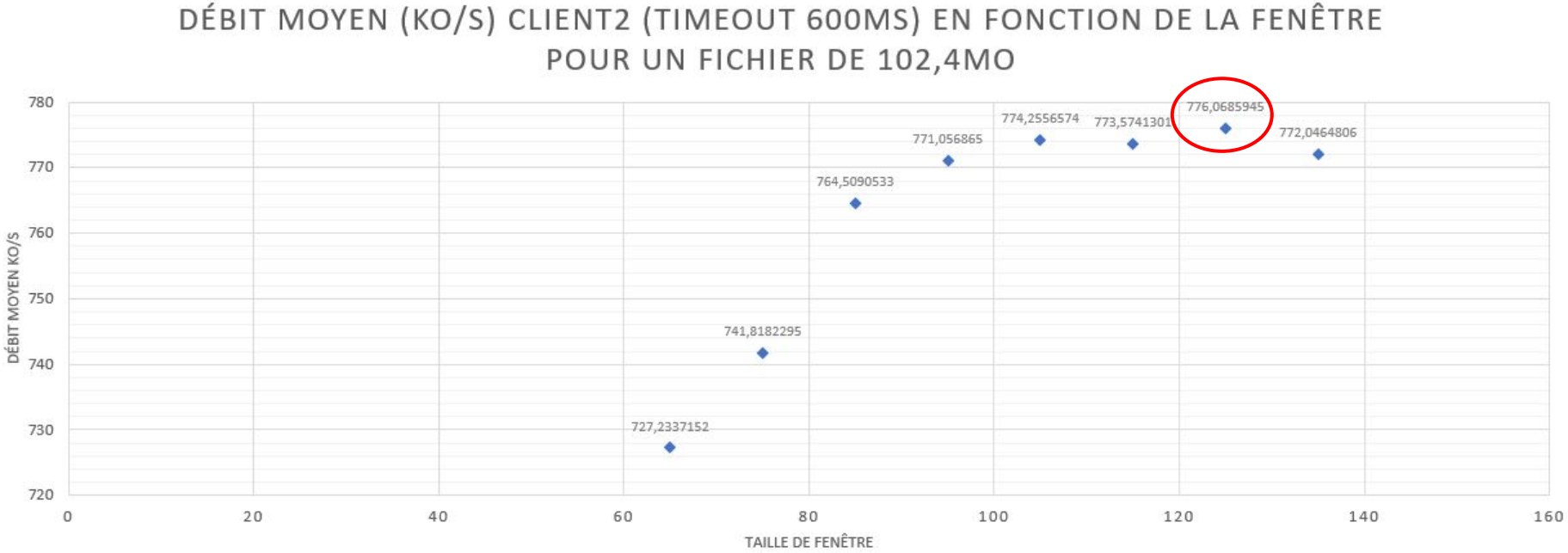
Scénario 1

2°) Recherche d'une configuration optimale



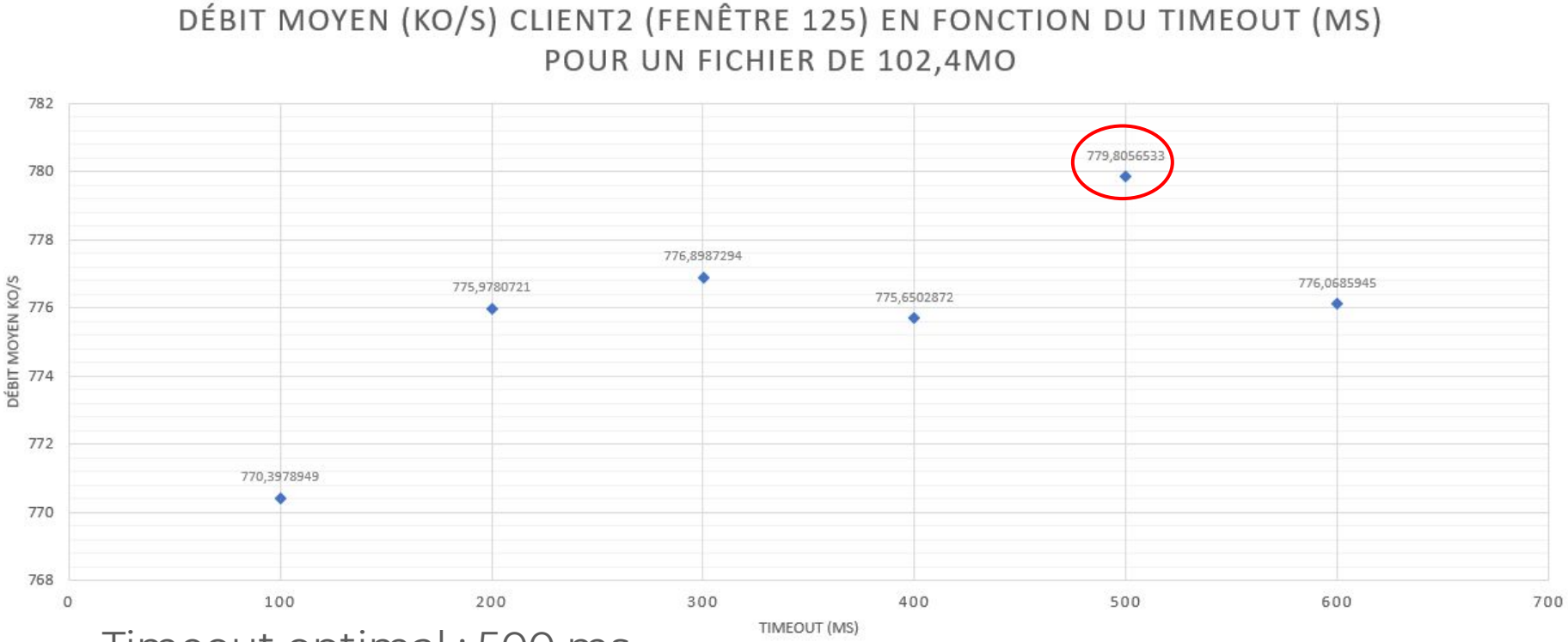
Timeout optimal : 150 ms

Scénario 2



Taille de fenêtre optimale : 125

Scénario 2



Timeout optimal : 500 ms

Scénario 3

Connexions (simultanées) de plusieurs clients de type client1 au serveur

- Le 3WH ainsi que l'ouverture de la nouvelle connexion reste synchrone
- Parallélisation complète de l'envoi des fichiers
- Test sur une pool de 5 clients lancés simultanément demandant chacun l'envoi d'un fichier de 2Gb

Limites et améliorations possibles

- Limite : chargement de l'entièreté du fichier à envoyer dans la map (on peut se retrouver limité par la RAM)

- Amélioration : recherche du goulot d'étranglement à l'aide de timers

```
START SEND:  2021-12-12 21:41:36.08915314 +0100 CET m=+7.950534325
END SEND:  2021-12-12 21:41:36.089194468 +0100 CET m=+7.950575723
START SEND:  2021-12-12 21:41:36.090270843 +0100 CET m=+7.951652038
END SEND:  2021-12-12 21:41:36.090309896 +0100 CET m=+7.951691081
END ACK:  2021-12-12 21:41:36.090409453 +0100 CET m=+7.951790649
START ACK:  2021-12-12 21:41:36.090416066 +0100 CET m=+7.951797261
END ACK:  2021-12-12 21:41:36.091016025 +0100 CET m=+7.952397210
START ACK:  2021-12-12 21:41:36.091023248 +0100 CET m=+7.952404463
END ACK:  2021-12-12 21:41:36.091192777 +0100 CET m=+7.952573972
START ACK:  2021-12-12 21:41:36.091195903 +0100 CET m=+7.952577088
START SEND:  2021-12-12 21:41:36.092261668 +0100 CET m=+7.953642883
END SEND:  2021-12-12 21:41:36.09228907 +0100 CET m=+7.953670255
START SEND:  2021-12-12 21:41:36.093357179 +0100 CET m=+7.954738364
END SEND:  2021-12-12 21:41:36.093378509 +0100 CET m=+7.954759694
```

Conclusion

- On a de **meilleures performances dans le scénario 2** par rapport au scénario 1
- On a des **temps d'exécutions similaires entre le scénario 1 et 3** grâce à la parallélisation mise en place
- **Points négatifs** : Notre code nécessite beaucoup de RAM pour envoyer de gros fichiers et on aurait pu rendre le code plus optimal en utilisant plus de go routines
- **Points positifs** : L'utilisation de go routines pour paralléliser les tâches, de fenêtres glissantes, la gestion des paquets perdus paramétrable avec la taille des fenêtres et les timeouts