Rapport Sprint

DAGUE Bastien, AL-FARAJ Samy, JOUINI Yassine, BROCHARD Vincent

Décembre 2025

Table des matières

1 Introduction

Pour cette première partie du rapport, nous allons expliquer certains noms et acronymes que nous utiliserons tout au long de celui-ci.

1.1 A-SABR

Adaptive Schedule-Aware Bundle Routing (A-SABR) est un framework open-source créé en Rust pour les algorithmes de routage DTN (Delay-Tolerant Networking). Les réseaux DTN sont faits pour les connexions intermittentes et planifiées, par exemple les satellites qui ont une fenêtre de connexion toutes les X heures ou minutes.

A-SABR va servir pour le routage de ce réseau en déterminant à quel moment le message doit partir pour avoir le chemin le plus court et soit le moins énergivore possible tout en évitant la saturation.

Code de A-SABR.

A-SABR possède également une interface graphique codée en Rust pour se répresenter le fonctionnement d'A-SABR. Elle simule le protocole UDP, TCP et BP.

Front-End de l'interface graphique.

Back-End de l'interface graphique

1.2 Hardy

Pour finir, nous introduisons ici le logiciel open-source Hardy. Il s'agit du logiciel global que l'on pourrait donner à un satellite ou tout objet devant communiquer par réseau DTN. Il utilise le protocole BPv7.

Code de Hardy.

2 Compte rendu de la réunion avec l'encadrant

La réunion a eu lieu le vendredi 12 décembre en début d'après midi au LIRMM où tout le groupe était présent. Mr De Jonckère nous a présenté le sujet en nous expliquant que tout n'était pas à faire, que c'était à nous de choisir ce qui nous semblait faisable.
Il nous a présenté sa bibliothèque ainsi qu'une petite démonstration de l'interface graphique, nous a exposé les avantages de l'utilisation de Rust, tout en nous faisant un cours introductif, et son utilisation dans le projet.

Nous avons dès lors exclu la partie no-std du sujet car trop dépendant du matériel.

Voici donc à l'issue de la réunion les 3 objectifs dont 2 prioritaires que nous nous sommes fixés :

— **Tests et Pipelines :**
Mise en place de tests pour la bibliothèque A-SABR et l'interface graphique.
Mise en place de Pipeline sur GitHub.

— **Modifications de l'interface graphique :**
Ajout d'un volet option dans l'interface graphique pour recharger l'interface avec un algorithme et un contact plan donnés.

— **Implémentation d'A-SABR dans Hardy :**
Ajout d'A-SABR dans le logiciel open-source de création de noeud DTN Hardy.

3 Cahier des charges

Pour ce projet, nous travaillerons dans le domaine des Delay-Tolerant Networks (DTN). Les réseaux DTN permettent la communication entre des nœuds très éloignés et dont la connexion est intermittente, comme c'est souvent le cas dans le domaine spatial en raison du mouvement des corps célestes. Le travail s'appuiera sur la bibliothèque A-SABR, développée au LIRMM. De plus, une interface graphique a été créée afin de faciliter l'utilisation de cette bibliothèque.

Notre objectif sera, dans un premier temps, de contribuer à ce projet en améliorant la testabilité de la bibliothèque et de son interface graphique, notamment par l'ajout de tests unitaires et d'intégration qui font actuellement défaut. Parallèlement, nous ajouterons une fonctionnalité à l'interface graphique pour permettre une configuration plus simple de l'outil et de ses paramètres.

Cette bibliothèque étant écrite en Rust, nous utiliserons ce langage pour nos développements. Côté environnement de travail, nous utiliserons l'IDE Visual Studio Code avec l'extension Rust Analyzer. Enfin, pour la gestion du projet, nous utiliserons GitHub, tandis que le développement s'effectuera sous Ubuntu via WSL (Windows Subsystem for Linux).

4 Première architecture du projet

Nous partons déjà d'une base de code fonctionnel, donc l'architecture du projet est déjà décidée et nous allons l'expliquer ci dessous.

4.1 A-SABR

Nous avons un premier dépôt contenant tout le code d'A-SABR, celui-ci va transformer un Contact Plan, en un graphe de contact contenant les opportunités de transmissions pour les sommets, et le temps de stockages pour les arêtes.
Il utilise pour cela une adapation de l'algorithme de Dijkstra pour trouver le chemin le plus rapide et l'algorithme de Yen pour trouver les meilleures routes sans boucles.
Une fois les tests et l'interface graphique modifiés c'est cette bibliothèque qu'on tentera d'intégrer dans le logiciel Hardy.

4.1.1 Exercices

Nous le précisons ici, mais dans le dépot A-SABR, il y a quelques exercices qui ont été mis de base pour un Hackaton et dont on va se servir pour prendre en main la librairie.

4.2 Dt-Chat

Comme dit dans l'introduction, une interface graphique existe pour se répresenter visuellement comment tout cela fonctionne. Elle est découpée en un back end et un front end.
Nous allons travailler sur le back end pour qu'il accepte de se recharger avec un algorithme et/ou un contact plan différent, et sur le front end pour un nouvel onglet option qui contiendra une option changement de contact plan par ajout de fichier ou changement d'algorithme parmi ceux proposés.

5 Organisation interne du groupe

Tout d'abord, la personne qui a été désignée comme chef du groupe est Bastien Dague. C'est lui qui s'occupera en grande partie de la bonne communication entre l'encadrant et le groupe.

Ensuite, pour ce qui est de la répartition des tâches, nous avons évalué au mieux le travail demandé pour chaque objectif afin de répartir le travail de manière efficace. Ainsi, quand la première phase d'apprentissage et de prise en main de Rust sera terminée, nous répartirons le groupe de cette manière :

— **Tests A-SABR et Pipelines :** Bastien Dague et Samy Al-Faraj
— **Modifications de l'interface graphique :** Vincent Brochard
— **Tests interface graphique :** Yassine Jouini
— **Implémentation d'A-SABR dans Hardy :** Tout le groupe

Concernant les outils d'organisation, nous utiliserons GitHub pour le suivi du projet. Le dépôt centralisera tous les sous-projets (A-SABR, interface graphique, Hardy) sous forme de sous-modules, qui redirigeront chacun vers des dépôts forkés des travaux de Monsieur De Jonckère.

Pour éviter tout problème de conflit, notre dépôt suivra la structure GitFlow. Chaque membre du groupe travaillera sur sa propre branche, et des merges réguliers vers la branche develop seront effectués. L'envoi sur la branche main ne se fera qu'après s'être assuré de la stabilité du projet. Par la suite, nous aurons la possibilité de soumettre des Pull Requests, avec l'accord de notre encadrant, pour contribuer aux dépôts originaux.

Enfin, pour la communication interne du groupe, nous utiliserons Discord. Pour le moment, les échanges avec notre encadrant se feront par e-mail, mais nous envisagerons de communiquer par messagerie instantanée lorsque le projet sera plus avancé.

6 Planning prévisionnel

Le projet se divisera en 3 phases :

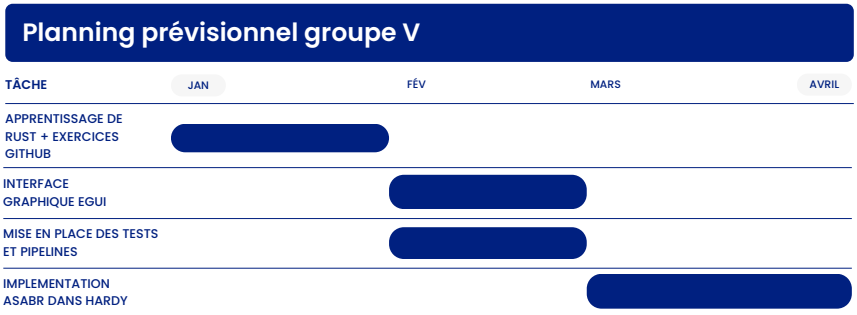
— **Phase 1 :** Apprentissage de Rust et travail sur les exercices du GitHub de Mr De Jonckère
— **Phase 2 :** Ajout d'une couverture de tests pour la librarie et l'interface graphique en parallèle, implémentation de fonctions de paramétrage pour l'interface graphique
— **Phase 3 :** Implémentation de la librarie A-SABR dans le logiciel libre Hardy

La première phase sera l'une des plus essentielles, car elle nous permettra de nous familiariser avec le langage Rust et la bibliothèque de M. De Jonckère. Grâce à ce travail en amont, nous pourrons commencer à travailler sur des bases solides pour la suite du projet.

Pour la seconde phase, notre priorité sera d'être aussi exhaustifs que possible dans nos tests, afin de garantir une meilleure testabilité présente et future. Nous visons également une utilisation plus flexible de l'interface graphique en lui implémentant un volet **Options**.

Enfin, lors de la dernière phase, la compréhension de la bibliothèque devrait être suffisante ; notre priorité se tournera donc vers l'analyse du logiciel Hardy pour tenter l'implémentation d'A-SABR.

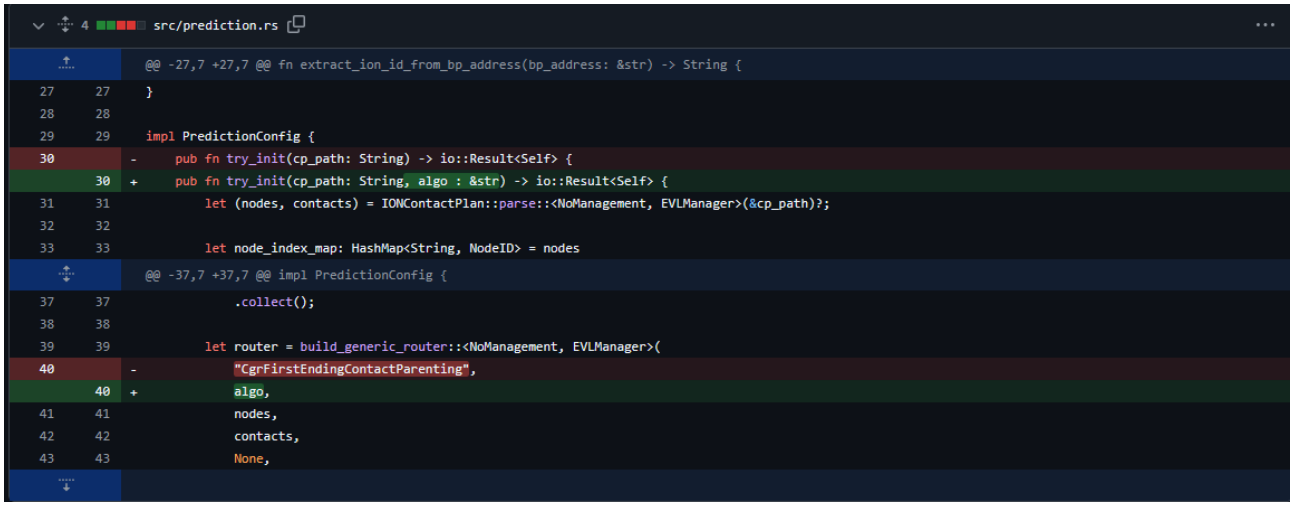
Planning prévu sous forme de diagramme de Gantt :



7 Prototype réalisé

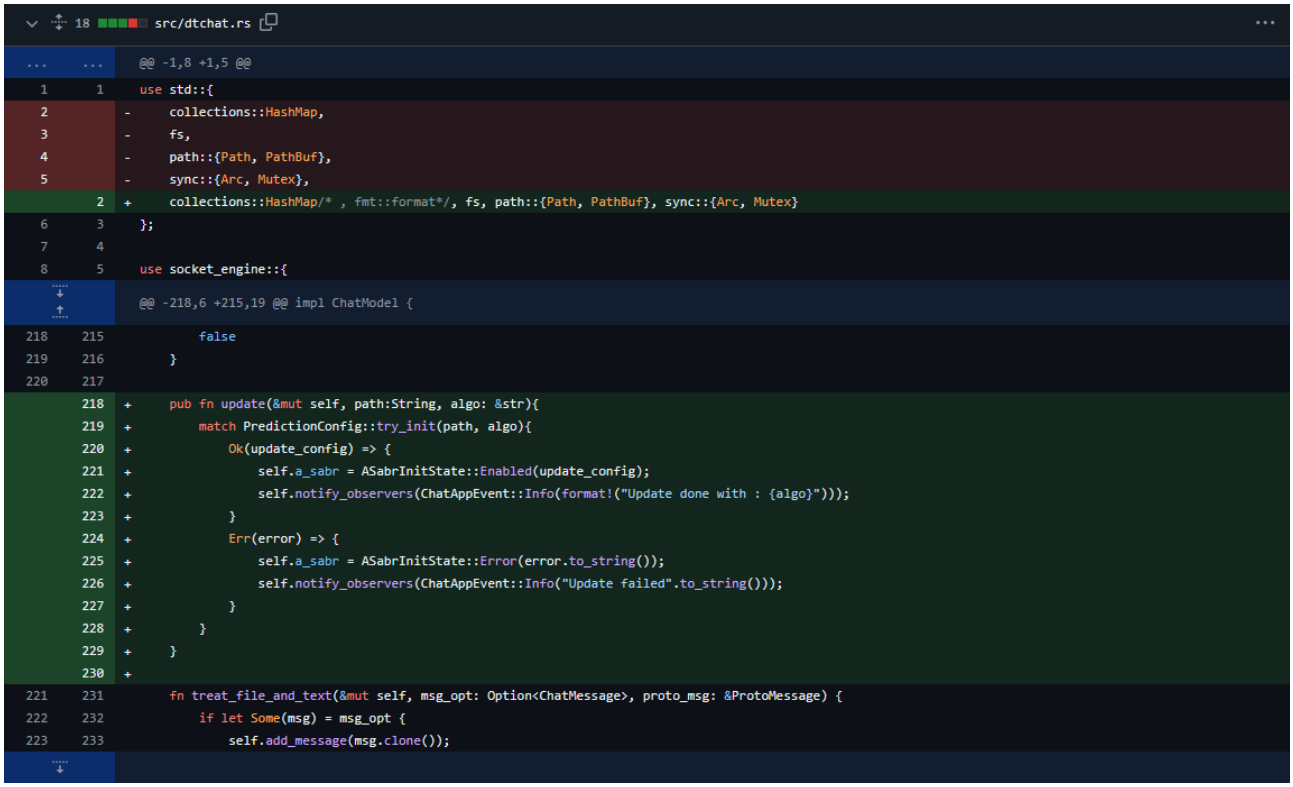
Pour le moment il s'agit surtout de prendre en main Rust par sa documentation, et la librairie par les exercices qu'elle propose, donc nous n'avons pas de prototype réalisé.

Néamoins, Vincent a proposé un petit changement sur le back end dans une branche dédiée, avec la modification d'une fonction pour qu'elle accepte un choix d'algo en plus du contact plan qu'elle accepte déjà et a crée une fonction update pour recharger l'interface avec un algo et/ou un contact plan différent.



Ici premièrement, nous modifions la signature de `try_init` pour qu'elle accepte en plus du contact plan habituel, un algorithme passé sous forme de `&str`, les algorithmes étant déjà notés tels quels en dur dans le code.

Nous corrigeons donc l'appel de la fonction associé `try_init` de `PredictionConfig`.
`PredictionConfig` est la structure contenant l'inistialisation des paramètres lors du lancement de l'interface graphique.



```
18 src/dtchat.rs
@@ -1,8 +1,5 @@
use std::{
-    collections::HashMap,
-    fs,
-    path::{Path, PathBuf},
-    sync::{Arc, Mutex},
+    collections::HashMap/* , fmt::format*/, fs, path::{Path, PathBuf}, sync::{Arc, Mutex}
};

use socket_engine::{
@@ -218,6 +215,19 @@ impl ChatModel {
        false
    }

+    pub fn update(&mut self, path:String, algo: &str){
+        match PredictionConfig::try_init(path, algo){
+            Ok(update_config) => {
+                self.a_sabr = ASabrInitState::Enabled(update_config);
+                self.notify_observers(ChatAppEvent::Info(format!("Update done with : {algo}")));
+            }
+            Err(error) => {
+                self.a_sabr = ASabrInitState::Error(error.to_string());
+                self.notify_observers(ChatAppEvent::Info("Update failed".to_string()));
+            }
+        }
+    }
+
    fn treat_file_and_text(&mut self, msg_opt: Option<ChatMessage>, proto_msg: &ProtoMessage) {
        if let Some(msg) = msg_opt {
            self.add_message(msg.clone());
```

Pour finir, dans la structure `ChatModel`, nous créons une fonction `Update` qui sera appelée à chaque modification du contact plan ou de l'algorithme dans le volet Options qui sera crée dans l'interface. Il s'agit simplement de faire un match en rappellant `PredictionConfig::try_init`, et si `Ok`, on reload avec les nouveaux résultats, sinon `Error` et on envoie l'erreur à l'interface sans crash.

Vincent a écrit ses fonctions en lisant attentivement le code et en retraçant quelle fonction est appelée en utilisant le contact plan.
Les modifications passent un `cargo check` et un `cargo build`

La prochaine étape consiste à valider ces modifications via le front-end.