Rapport de Projet : C-WildWater

Analyse et Visualisation de Réseau Hydraulique

Pré-Ing 2 – 2025-2026

Équipe Projet :

Myriam Bensaid
Sheryne Ouarghi
Matthieu Vannereau

Date : 19 décembre 2025

Table des matières

1 Introduction

Ce projet, intitulé **C-WildWater**, nous a plongés au cœur du traitement de données massives. Notre mission était de concevoir une chaîne logicielle capable d'analyser un réseau de distribution d'eau complexe en France. Le défi n'était pas seulement de lire un fichier CSV, mais de le faire intelligemment : extraire des statistiques, repérer des fuites et visualiser tout cela de manière claire, le tout en C et Shell.

Nous avons dû jongler entre la gestion de la mémoire, l'implémentation de structures de données performantes (AVL) et l'automatisation des tâches, ce qui nous a permis de consolider nos compétences techniques.

2 Organisation de l'Équipe

Pour mener à bien ce projet, nous avons réparti les responsabilités selon nos points forts respectifs, tout en gardant une vision d'ensemble sur le code :

— **Myriam** a pris en charge l'architecture globale du code. Elle s'est occupée de développer les fonctions utilitaires essentielles, de gérer les "merges" Git pour garder un historique propre, et a travaillé sur l'affichage final dans le terminal. Elle a également piloté les tentatives d'optimisation du code.

— **Sheryne** s'est concentrée sur la visualisation des données. C'est elle qui a maîtrisé Gnuplot pour transformer nos fichiers CSV bruts en graphiques lisibles et pertinents, assurant ainsi que nos résultats soient interprétables visuellement.

— **Matthieu** s'est attaqué à la logique métier la plus complexe. Il a développé l'algorithme de calcul des fuites, la détection des "pires tronçons" du réseau, et a préparé l'implémentation du bonus (les histogrammes empilés).

3 Architecture Technique

Notre solution repose sur trois piliers :

1. **Le Chef d'Orchestre (Shell) :** Le script ‘myScript.sh‘ automatise tout. Il compile, nettoie l'environnement, lance le programme C avec les bons paramètres et déclenche Gnuplot.
2. **Le Moteur (C) :** C'est ici que la magie opère. Le programme lit le CSV, construit un arbre AVL en mémoire pour stocker les stations, et effectue les calculs (sommes, différences, détections).
3. **La Vitrine (Gnuplot) :** Une fois les données traitées, Gnuplot génère les images PNG que nous analysons.

4 Choix des Structures de Données

Très vite, nous avons compris qu'une liste chaînée simple serait trop lente vu la taille du fichier. Nous avons donc opté pour un **Arbre AVL**.

```
typedef struct Station {
    unsigned int id;            // ID unique
    long capacity;              // Capacite
    long load;                  // Charge
```

```
    struct Station *left;       // Equilibrage AVL
    struct Station *right;
    int height;
    // ...
} Station;
```

Listing 1 – Notre structure Station

Ce choix nous garantit que même dans le pire des cas, retrouver une station pour mettre à jour ses données se fait très rapidement (complexité logarithmique).

5 Difficultés et Apprentissages

Ce projet ne s'est pas fait sans heurts. Voici les principaux obstacles que nous avons dû surmonter :

5.1 Le "Cauchemar" du Parsing

Au début, nous avons sous-estimé la complexité du fichier CSV. Entre les séparateurs parfois ambigus et les formats de données variables, nous avons eu plusieurs "Segmentation Faults". Myriam a dû retravailler les fonctions de lecture pour les rendre plus robustes et tolérantes aux irrégularités du fichier.

5.2 La Logique des AVL

Comprendre la théorie des AVL est une chose, l'implémenter en est une autre. La gestion des rotations (simples et doubles) nous a donné du fil à retordre. Nous avons passé beaucoup de temps à dessiner des arbres sur papier pour comprendre pourquoi certains nœuds se perdaient lors des rééquilibrages.

5.3 L'Intégration du Bonus (Gnuplot)

Pour le bonus géré par Matthieu, il ne suffisait pas de sortir des chiffres. Il fallait formater les données d'une façon très précise pour que Sheryne puisse configurer Gnuplot en mode "histogramme empilé". Faire communiquer correctement la logique C avec les attentes du script Gnuplot a demandé plusieurs itérations de tests.

6 Optimisation et Crédits

Dans une optique de performance, nous avons exploré le multi-threading. Comme le montrent les fichiers `multiThreaded.c`, nous avons utilisé des `mutex` pour sécuriser l'écriture dans les fichiers lors de traitements parallèles.

Crédits : L'idée et la base technique de cette implémentation multi-threadée proviennent du travail de **Tiago Charette**. Nous nous sommes inspirés de son dépôt GitHub [1] que nous avons adapté à notre structure.

1. https://github.com/TogExe/MultiThreaded

7 Résultats Visuels

Voici les fruits de notre travail, générés automatiquement par notre chaîne de traitement.

7.1 Capacités (Mode Max)

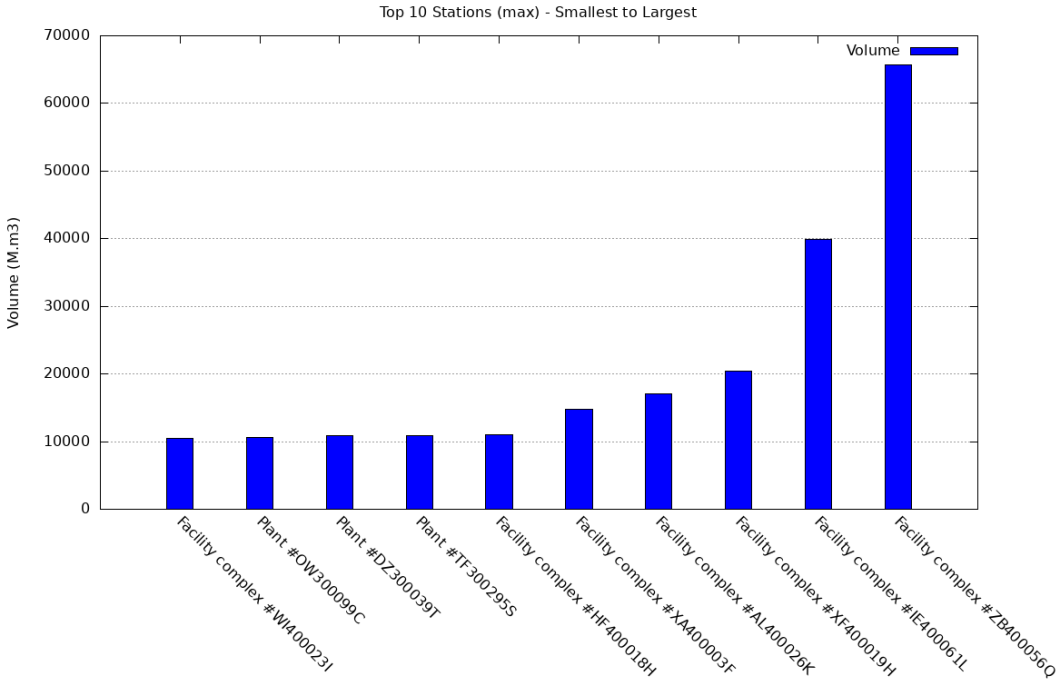

FIGURE 1 – Les 10 stations les plus capacitaires

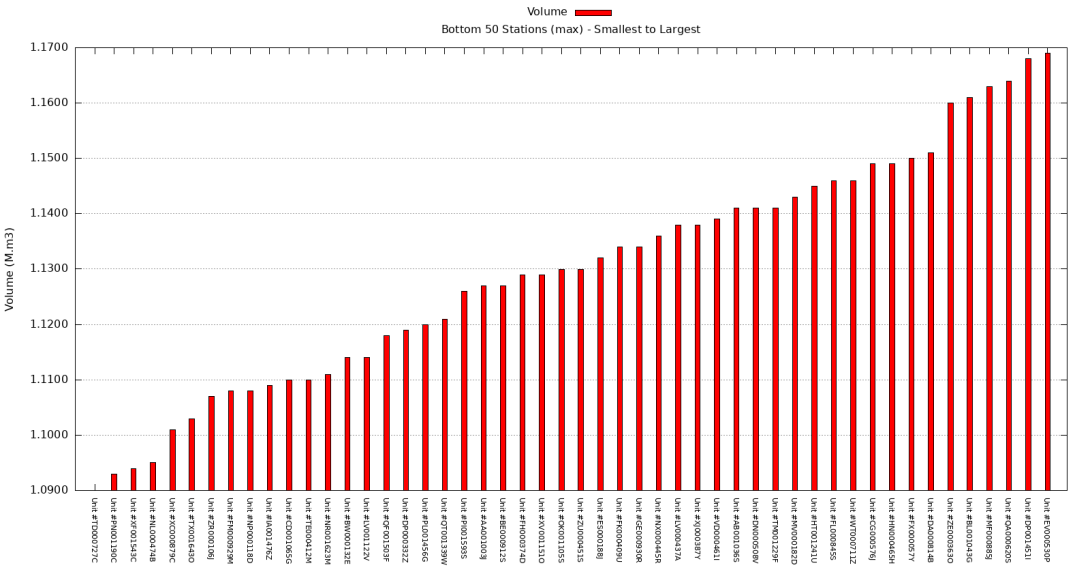

FIGURE 2 – Les 50 stations avec les plus petites capacités

7.2 Flux Réels (Mode Real)

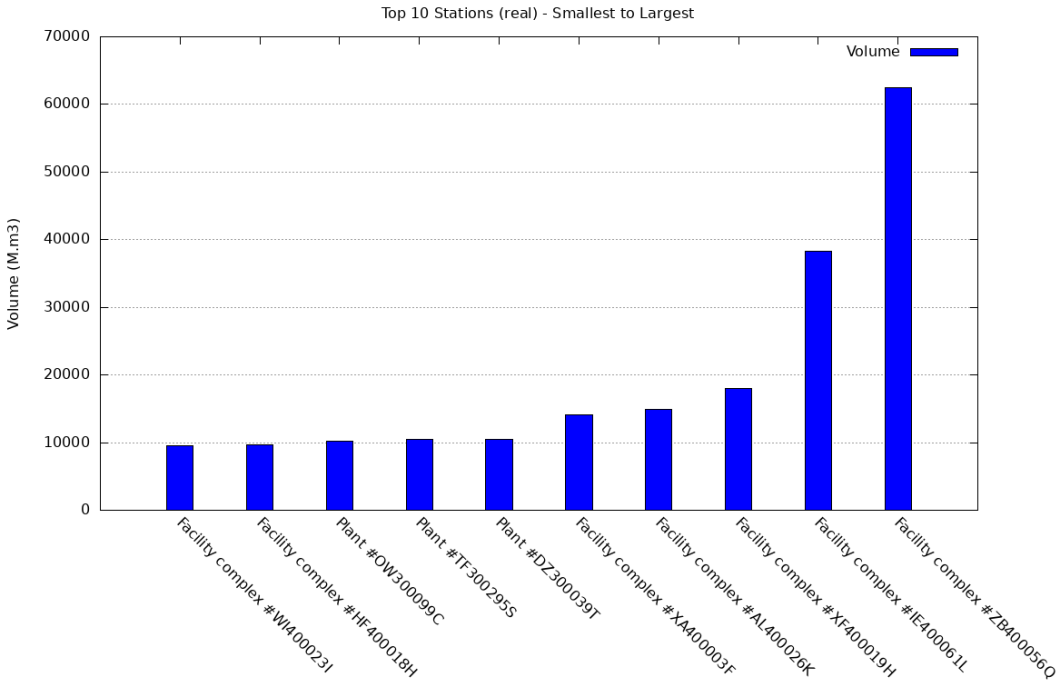

FIGURE 3 – Top 10 des débits réels

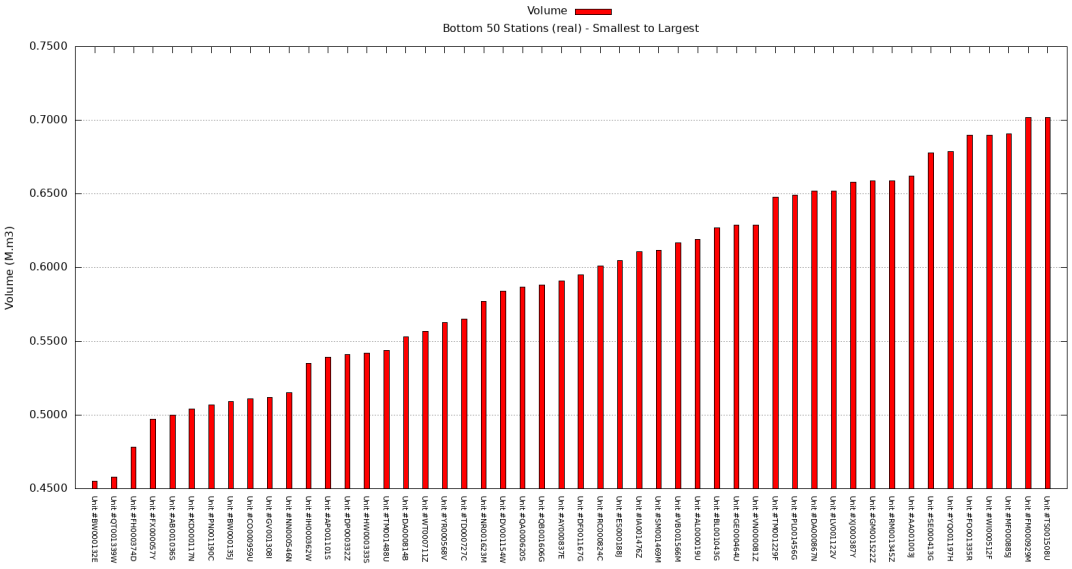

FIGURE 4 – Les 50 plus petits débits

7.3 Sources (Mode Src)

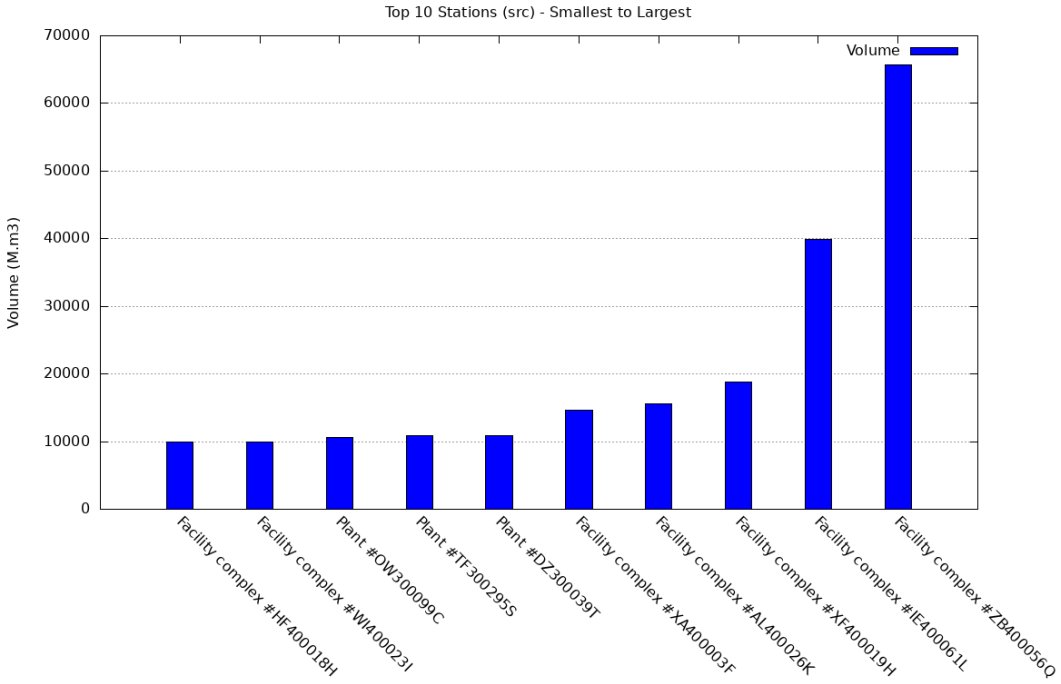

FIGURE 5 – Top 10 des sources d'eau

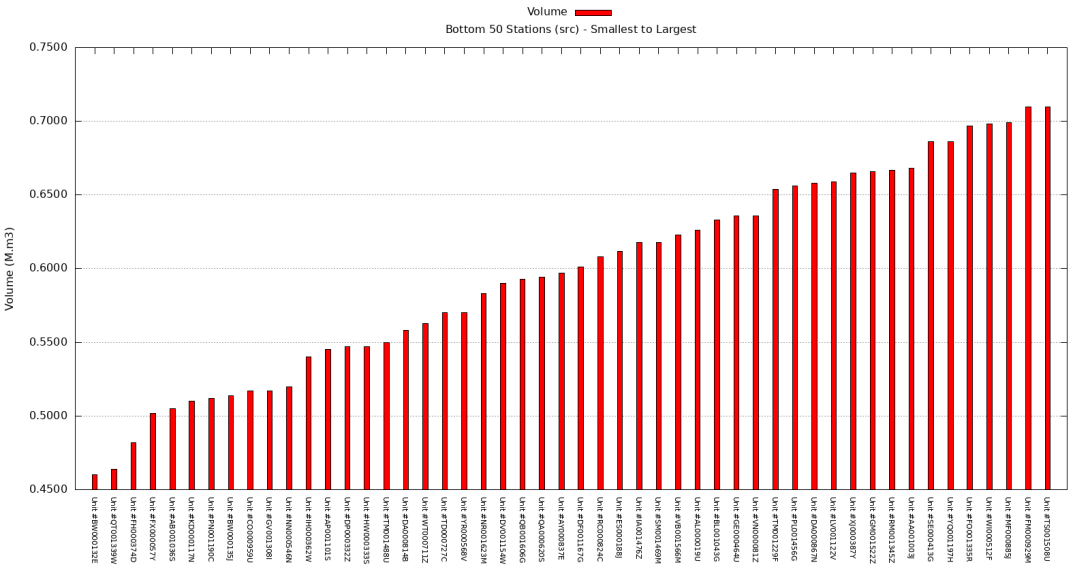

FIGURE 6 – Les 50 plus petites sources

7.4 Analyse Avancée (Bonus)

Ces graphiques sont particulièrement intéressants car ils montrent la répartition : en vert ce qui est vide, en rouge les pertes (fuites ou surconsommation), et en bleu le flux normal.

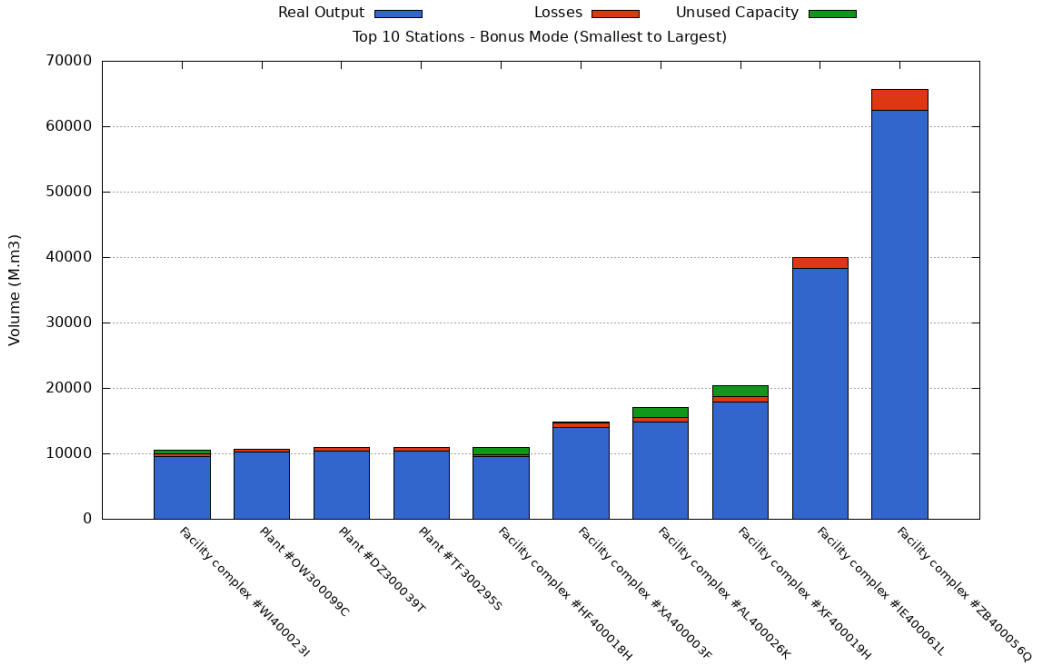

FIGURE 7 – Détail du Top 10 (avec pertes et capacité libre)

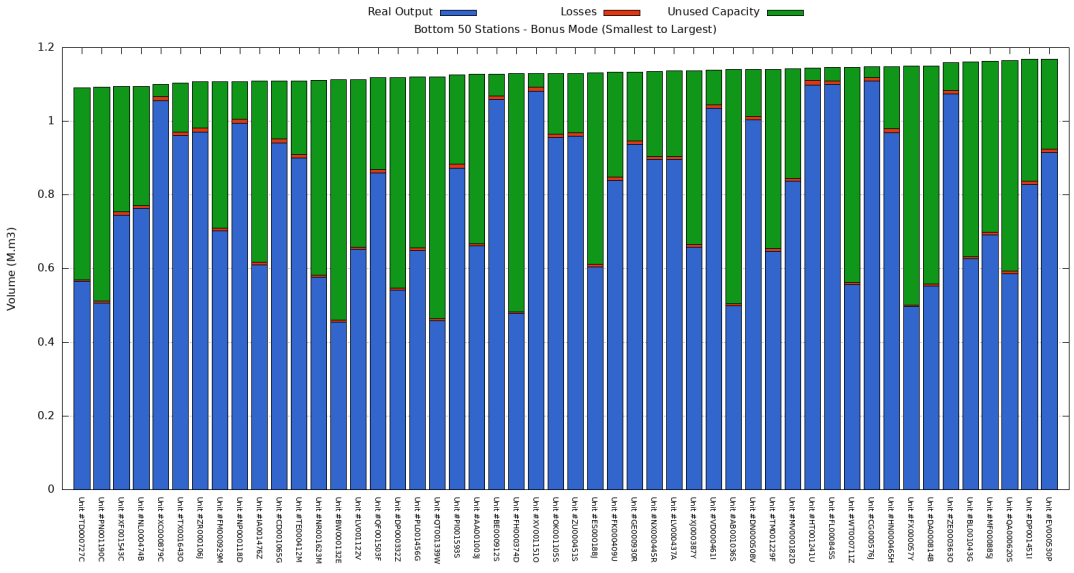

FIGURE 8 – Détail des 50 plus petites stations

8 Conclusion

Au final, C-WildWater a été un projet très formateur. Nous avons appris à travailler en équipe sur une base de code commune, à gérer les conflits de fusion, et surtout à déboguer des problèmes de mémoire complexes. La satisfaction de voir les graphiques s'générer correctement à la fin valait bien les heures passées à corriger les pointeurs NULL !