Sensor Monitoring System

1. Contexte et Structure

Vue d'ensemble

Système de monitoring environnemental distribué permettant la collecte, le stockage et la visualisation en temps réel de données environnementales (température, pression atmosphérique, humidité) via un réseau Ethernet local.

2. Objectifs et Contributions

Objectif principal

Développer un système de monitoring complet.

Rôle et Responsabilités

Développeur Système (Projet autodidact - 100% des contributions)

- **Architecture système** : Conception d'une architecture distribuée publisher/subscriber via MQTT
- **Développement embarqué** : Programmation C++ sur ESP32 avec gestion réseau Ethernet
- **Backend système** : Développement d'un serveur C multi-thread avec persistance SQLite3
- **Automatisation** : Scripts Bash pour la maintenance
- **Configuration** : Système centralisé via configuration TOML

Compétences démontrées

- **Programmation système bas niveau** : C/C++
- **Protocoles réseau** : TCP/IP, MQTT (QoS 1), Ethernet
- **Bases de données** : SQLite3 avec optimisations : WAL, indexation, vacuum incrémental
- **DevOps** : Makefile, scripts de déploiement, gestion de dépendances
- **Systèmes embarqués** : ESP32, capteurs I2C (BME280), SPI (W5500)

3. Environnement et Contraintes

Environnement technique

Hardware

- MCU : ESP32
- Capteur : BME280 (I2C 0x76/0x77)
- Network : W5500 Ethernet Controller (SPI)
- Serveur : Linux Arch

Réseau

- Configuration IP statique (192.168.69.0/24)
- Communication directe ESP32 - Serveur
- Latence cible : <100ms

Contraintes techniques

Embarqué

- Mémoire limitée : RAM 520KB, Flash 4MB
- Pas de système de fichiers : configuration hard-coded
- Mode économie d'énergie : BME280 en FORCED mode
- Gestion reconnexion : max 3 échecs avant reset complet

Serveur

- Performance : Insertion BDD <10ms par mesure
- Stockage : Rotation des données
- Disponibilité : Daemon sans supervision externe

Système

- Zero downtime : Reconnexion automatique MQTT
- Intégrité données : Transactions SQLite sécurisées
- Observabilité : Logs structurés
- Maintenance : Scripting

4. Technologies et Outils

Stack technique

Embarqué (ESP32)

Framework : Arduino Core
Langage : C++
Build : PlatformIO
Libs : Adafruit_BME280, Ethernet, PubSubClient, ArduinoJson

Backend (Serveur Linux)

Langage : C
Compilateur : GCC avec flags -Wall -Wextra -O2
Libs : paho-mqtt3c, json-c, sqlite3, tomlc99
Build : Makefile avec détection dépendances

Infrastructure

OS : Linux (Arch)
Broker MQTT : Mosquitto
Shell : Zsh
DB : SQLite 3 (mode WAL)

Choix d'implémentation

Protocole MQTT

- QoS 1 (at least once) : Équilibre fiabilité/performance
- Keepalive 60s : Détection déconnexion rapide
- Clean session : État non persistant côté broker

Optimisations SQLite

```
PRAGMA journal_mode=WAL;          -- Write-Ahead Logging
PRAGMA synchronous=NORMAL;        -- Fsync réduit
PRAGMA temp_store=MEMORY;         -- Temp en RAM
PRAGMA auto_vacuum=INCREMENTAL;   -- Défragmentation contrôlée
```

Gestion d'erreurs

- ESP32 : Reset système après 3 échecs MQTT consécutifs
- Serveur : Reconnexion automatique avec backoff (15s)
- Réseau : Validation avant le build via network.sh

5. Résultats et Livrables

Fonctionnalités implémentées

Acquisition données

- Échantillonnage BME280 à une fréquence fixée
- Précision : ~0.5°C, ~1hPa, ~3%RH
- Format JSON standardisé

Communication réseau

- Publisher MQTT fiable (QoS 1)
- Gestion reconnexion automatique
- Diagnostic réseau en temps réel

Stockage et persistence

- Base SQLite optimisée (WAL mode)
- Index sur timestamp pour requêtes rapides
- Cleanup automatique par batch (2000 lignes)

Monitoring et alertes

- Détection dépassement seuils en temps réel
- Alertes sur les passages de seuils
- Logs horodatés UTC

Automatisation

- Build reproductible via Makefile
- Scripts de maintenance
- Configuration centralisée TOML

Livrables

Code source

- 5 fichiers C/C++ documentés (~1200 lignes)
- 2 scripts Bash de production
- Makefile avec gestion dépendances

Documentation

- README.md avec quickstart
- PDF de documentation
- Configuration annotée (config.toml)

Infrastructure

- Système déployable en <5 min
- Scripts de validation réseau
- Procédure de maintenance automatisée

Captures d'écrans

Affichage des envois depuis l'ESP (à droite) et des message reçus par MQTT sur le Serveur (à gauche) :

```
=== Message reçu ===
Date, Heure : 2026-01-02 15:48:21
Topic : esp32/env
Données parsées :
 - Température : 17.0 °C
 - Pression : 985.9 hPa
 - Humidité : 44 %
=== Message enregistré ===

=== Message reçu ===
Date, Heure : 2026-01-02 15:48:26
Topic : esp32/env
Données parsées :
 - Température : 17.0 °C
 - Pression : 985.9 hPa
 - Humidité : 44 %
=== Message enregistré ===
```

```
=== Envoi ===
{"temperature":17.1,"pression":985.7,"humidite":44.3}
=== Envoyé ===

=== Envoi ===
{"temperature":17.1,"pression":985.8,"humidite":44.2}
=== Envoyé ===

=== Envoi ===
{"temperature":17.1,"pression":985.8,"humidite":44.4}
=== Envoyé ===
```

Configuration et Vérification lors du démarrage du Serveur :

```
=== Subscriber MQTT ===

=== Configuration chargée ===
Racine projet : /home/Arch/Projets/sensor-monitoring-system

[MQTT]
  Broker : tcp://localhost:1883
  Topic : esp32/env
  Client ID : UnixSubscriber
  QoS : 1
  Keepalive : 60 s

[Database]
  Path : data/donnees_esp32.db
  Rétention : 3 heures
  Batch cleanup : 2000

[Logging]
  Alertes : data/alertes.log
  Cleanup : scripts/cleanbd.log

[Seuils]
  Température : 17.0°C - 24.0°C
  Pression : 980.0 - 1030.0 hPa
  Humidité : 40% - 70%

[Affichage]
  Messages : désactivé

============================

Heure système UTC : 2026-01-02 15:45:05

Seuils d'alerte configurés:
  Température : 17.0°C à 24.0°C
  Pression : 980.0 à 1030.0 hPa
  Humidité : 40% à 70%
Base de données prête (/home/Arch/Projets/sensor-monitoring-system/data/donnees_esp32.db)
Connexion au broker MQTT (tcp://localhost:1883)...
  Connecté au broker
Abonnement à esp32/env
  Abonné
```

```
Lancement du serveur...
Configuration chargée :
  Interface serveur : enp0s25
  IP serveur        : 192.168.69.1
  IP ESP32          : 192.168.69.2
  Port MQTT         : 1883

Vérification interface réseau...
  Interface enp0s25 existe
Vérification adresse IP...
  IP correcte : 192.168.69.1
Test connectivité ESP32...
  ESP32 joignable (192.168.69.2)
Vérification broker MQTT...
  Mosquitto actif
  Mosquitto écoute sur port 1883
Test connexion broker MQTT...
  Connexion MQTT fonctionnelle

Configuration réseau OK
```

Détection d'une alerte sur la température trop basse :

```
=== Message reçu ===
Date, Heure : 2026-01-02 15:53:21
Topic : esp32/env
Données parsées :
 - Température : 16.9 °C
 - Pression : 985.8 hPa
 - Humidité : 43 %
[2026-01-02 15:53:21] ALERTE : Température trop basse (16.9°C < 17.0°C)
=== Message enregistré ===
```

Lecture du fichier de journalisation des alertes :

```
❯ cat data/alertes.log
[2026-01-02 15:53:21] ALERTE : Température trop basse (16.9°C < 17.0°C)
[2026-01-02 15:55:06] Température revenue à la normale (17.2°C)
```

Lecture des données récoltées dans la BDD SQLite 3 :

```
❯ sqlite3 data/donnees_esp32.db
SQLite version 3.51.1 2025-11-28 17:28:25
Enter ".help" for usage hints.
sqlite> SELECT * FROM mesures;
1|2026-01-02 15:45:11|17.1|985.8|44
2|2026-01-02 15:45:16|17.1|985.8|44
3|2026-01-02 15:45:21|17.1|985.8|44
4|2026-01-02 15:45:26|17.1|985.8|44
5|2026-01-02 15:45:31|17.1|985.8|44
6|2026-01-02 15:45:36|17.1|985.8|44
7|2026-01-02 15:45:41|17.1|985.8|44
8|2026-01-02 15:45:46|17.1|985.7|44
9|2026-01-02 15:45:51|17.1|985.8|44
10|2026-01-02 15:45:56|17.1|985.8|44
11|2026-01-02 15:46:01|17.1|985.8|44
12|2026-01-02 15:46:06|17.1|985.8|44
13|2026-01-02 15:46:11|17.1|985.8|44
14|2026-01-02 15:46:16|17.1|985.8|44
15|2026-01-02 15:46:21|17.1|985.8|44
16|2026-01-02 15:46:26|17.1|985.8|43
17|2026-01-02 15:46:31|17.1|985.8|44
18|2026-01-02 15:46:36|17.1|985.8|44
19|2026-01-02 15:46:41|17.1|985.8|44
20|2026-01-02 15:46:46|17.1|985.8|44
sqlite> .quit
```

Évolutions futures

Court terme

- Interface GUI PyQt5 pour visualisation temps réel
- Dashboard web et/ou android

Moyen terme

- Support multi-capteurs
- Ecran de visualisation système embarqué

Long terme

- Mode WiFi

Conclusion

Ce projet démontre la capacité à concevoir et implémenter un système distribué complet, de la programmation embarquée bas niveau à l'automatisation système Linux. L'accent mis sur la robustesse (gestion d'erreurs, reconnexion automatique), la performance (optimisations DB, mode WAL) et la maintenabilité (configuration centralisée, scripts idempotents) reflète une approche professionnelle du développement système.

Points forts techniques :

- Maîtrise de la stack C/C++ système
- Compréhension des protocoles réseau (TCP/IP, MQTT)
- Optimisation base de données relationnelle
- Culture DevOps (automatisation, reproductibilité)

https://github.com/clement-dev-sys
https://github.com/clement-dev-sys/sensor-monitoring-system