TP 1 : Développement d'une application de collecte de données et de supervision via sockets

⏱ 3h
Réseaux de capteurs
Mis à jour le 04/11/2025

🎯 Objectifs

- Se familiariser avec la programmation de sockets TCP/UDP
- Comprendre le modèle client-serveur

🔧 Matériel nécessaire

- Ordinateur avec Python 3.x installé
- Accès au réseau local

> **Important**
>
> L'évaluation de ce TP se fait pendant la séance. Pensez à faire valider régulièrement votre avancement auprès de l'enseignant.

Description de l'application de supervision

On souhaite collecter les tensions et courants d'un ensemble de capteurs placés sur un système de type micro-grid électrique. Ces données servent ensuite aux autres applications pour planifier, contrôler et optimiser les opérations du micro-grid. Afin de collecter ces données, nous allons développer une application *Python* utilisant les sockets pour récupérer les données des capteurs.

> **Note**
>
> Cette démarche est applicable à d'autres types de données et systèmes nécessitant une collecte et une supervision en temps réel.

L'application à développer collecte des données simulées provenant de deux capteurs :

- un capteur de tension qui envoie ses données via UDP ;
- un capteur de courant qui envoie ses données via TCP.

Chacun des capteurs génère des données aléatoires dans des plages spécifiques :

- Tension : 220 ± 10V ;
- Courant : 5 ± 1A.

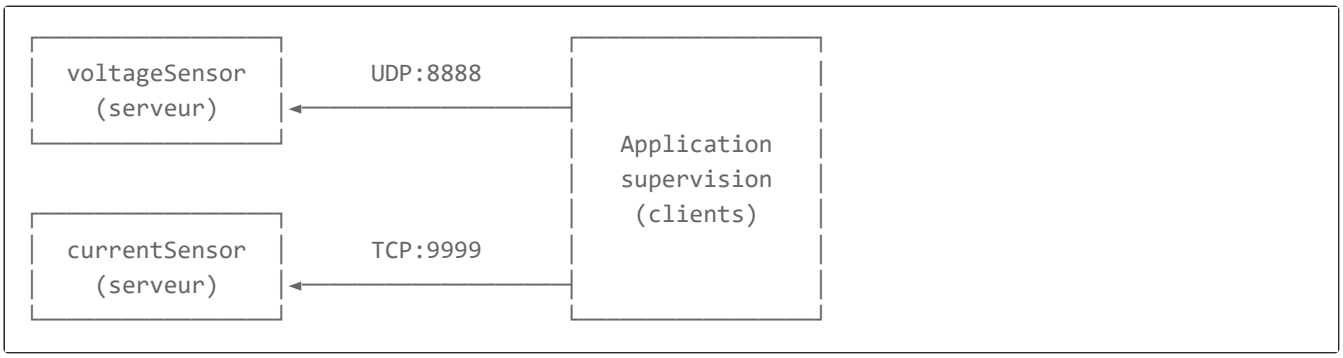
Une fois générées, les données sont stockées dans un fichier `json`.

L'application doit récupérer les données, les afficher dans la console et permettre à l'utilisateur de visualiser les graphiques correspondants grâce à la bibliothèque *matplotlib*.

> **Note**
>
> Des squelettes de code ainsi que des exemples de code socket client/serveur sont disponibles sur Arche.

> **Tip**
>
> Pensez à tester vos codes dès que vous implémentez une nouvelle fonction pour faciliter le débogage.

Architecture de l'application



Partie 1 : Modules d'acquisition de données

Dans cette première partie, nous allons programmer les deux modules de simulation de capteurs.

La classe `Sensor` définie dans le fichier `sensor.py` fournit une structure de base pour les capteurs. Elle possède un attribut `nb` qui indique le nombre de valeurs à générer, ainsi qu'un attribut `mesures` qui stocke les valeurs générées dans un dictionnaire.

Q1 (2 pts). Créez un fichier `voltageSensor.py` contenant la définition de la classe `voltageSensor` qui hérite de la classe `Sensor`. Cette classe doit inclure une méthode `generate()` qui :

- génère `nb` valeurs aléatoires de tension tirées uniformément entre 210V et 230V ;
- génère `nb` instants de mesure (en secondes) selon la formule $i + \text{random}(0, 1), i \in [0, nb - 1]$;
- stocke ces mesures dans le dictionnaire `mesures` dont les clés sont les instants et les valeurs sont les tensions correspondantes ;
- stocke le dictionnaire `mesures` dans un fichier `json` nommé `voltage.json`.

> **Note**
>
> Utilisez la fonction `dump()` du module `json` pour écrire un dictionnaire dans un fichier `json`.

> **Tip**
>
> Ajouter le paramètre `indent=4` à la fonction `dump()` permet d'obtenir un fichier `json` plus lisible.

Les commandes

```
>>> vs = voltageSensor(5)
>>> vs.generate()
```

génèrent par exemple le fichier `voltage.json` suivant :

```
{
    "0.1567103252601113": 225.55530087394834,
    "1.2164839663235272": 224.70993120869522,
    "2.8837368183033956": 215.0099734534407,
    "3.129765176593175": 217.15948563534192,
    "4.300469195840086": 215.3837650640686
}
```

Q2 (3 pts). Implémentez une méthode `serve()` qui crée un serveur UDP en écoute sur le port `8888`. Ce serveur doit générer les mesures de tension et les envoyer au client dès qu'une requête est reçue.

> **Note**
>
> Utilisez la fonction `dumps()` du module `json` pour convertir un dictionnaire en chaîne de caractères JSON.

> **Note**
>
> Utilisez la méthode `encode()` pour encoder une chaîne de caractères au format `utf-8` avant de l'envoyer via le socket.

Testez votre serveur UDP avec le script `UDPclient.py` fourni.

> **Warning**
>
> Vous devrez lancer plusieurs consoles *(shell)* pour exécuter les différents scripts (serveur et client).

Vous devriez obtenir une sortie similaire à celle-ci dans la console du client :

```
Input lowercase sentence: test
{
    "0.1567103252601113": 225.55530087394834,
    "1.2164839663235272": 224.70993120869522,
    "2.8837368183033956": 215.0099734534407,
    "3.129765176593175": 217.15948563534192,
    "4.300469195840086": 215.3837650640686
}
```

Q3 (4 pts). Créez un fichier `currentSensor.py` contenant la définition de la classe `currentSensor` permettant de simuler un capteur de courant. Les instants de mesure doivent être **identiques** à ceux du capteur de tension. Ce capteur doit servir les données via un serveur TCP en écoute sur le port `9999`.

Testez votre serveur TCP avec le script `TCPclient.py` fourni.

Partie 2 : Module de supervision

Dans cette seconde partie, nous allons développer le module de supervision qui récupère les données des deux capteurs, les affiche dans la console et permet de les visualiser graphiquement.

Q4 (2 pts). Dans le fichier `app.py`, coder la fonction `getVoltage()` qui :

- crée un client UDP vers un serveur écoutant sur le port `8888` ;
- récupère les mesures du capteur de tension ;
- retourne un dictionnaire qui contient les données reçues.

> **Note**
>
> Utilisez la fonction `loads()` du module `json` pour convertir une chaîne de caractères JSON en dictionnaire Python.

Q5 (2 pts). Dans le fichier `app.py`, coder la fonction `getCurrent()` qui retourne un dictionnaire contenant les mesures du capteur de courant.

Le fichier `app.py` contient déjà deux fonctions :

- `displayData()` permet d'afficher le contenu d'un dictionnaire de mesures donné en paramètre dans la console ;
- `menu()` affiche dans la console un menu demandant à l'utilisateur une opération à réaliser

```
1 - Collecter données du capteur de tension
2 - Collecter données du capteur de courant
3 - Visualiser les données
```

Q6 (1 pt). Dans le fichier `app.py`, coder la fonction `dump()` qui :

- prend deux arguments en entrée :
  - `filename` : le nom du fichier
  - `mesures` : un dictionnaire de mesures
- sauvegarde les données du dictionnaires `mesures` dans le fichier texte nommé `filename` selon le format suivant :

```
0.1567103252601113 225.55530087394834
1.2164839663235272 224.70993120869522
2.8837368183033956 215.0099734534407
3.129765176593175 217.15948563534192
4.300469195840086 215.3837650640686
```

Vous pourrez tester votre fonction avec le code suivant :

```
>>> mesures = { "1": 10, "2": 20, "3": 30 }
>>> dump("test.txt", mesures)
```

Le fichier `test.txt` doit contenir :

```
1 10
2 20
3 30
```

Q7 (1pt). Complétez la fonction `menu()` et testez votre application de supervision en ayant exécuté au préalable les serveurs des capteurs de tension et de courant sur la même machine *(localhost)*. L'application doit afficher le menu et permettre d'afficher les données récupérées dans la console et les sauvegarder dans des fichiers texte `voltage.txt` et `current.txt`.

Q8 (3pts). Modifiez votre application de supervision pour récupérer les données de deux capteurs dont les serveurs servent sur deux machines différentes de la machine locale. Après avoir lancé **Wireshark**, testez votre application, analysez les échanges de paquets entre les machines et expliquez-les à l'enseignant.

> **Tip**

> Utilisez des filtres d'affichage dans Wireshark pour ne visualiser que les paquets UDP et TCP échangés entre les machines.

Q9 (2pts). Dans le fichier `app.py`, coder la fonction `visualize()` qui utilise la bibliothèque *matplotlib* pour afficher trois graphiques à partir des dictionnaires collectés :

- un graphique de la tension en fonction du temps ;
- un graphique du courant en fonction du temps ;
- un graphique de la puissance instantanée en fonction du temps.

> ⚠ **Warning**
>
> Assurez-vous que les capteurs soient paramétrés avec le même `nb` pour pouvoir calculer la puissance instantanée.

Thomas Mongaillard
thomas.mongaillard@univ-lorraine.fr

Supports de TP/TD à l'ENSEM