UNIVERSITÉ D'ÉTÉ

Smart Windows

Raed Abdennadher – Orphée Antoniadis – Steven Liatti

Table des matières

Introduction

Dans le cadre de notre Université d'été, nous avons eu à développer un projet sur l'Internet des objets. Pour ce faire, nous avons eu à disposition une carte Waspmote ainsi que plusieurs modules de communication se connectant sur la carte. Nous avons rapidement imaginé une application dans la domotique. L'idée était de mettre en place tout un système d'ouverture automatique des fenêtres et des stores pour l'aération et la luminosité d'une pièce.

Nous avons donc imaginé placer un SensorTag servant de capteur de température à l'extérieur et un autre à l'intérieur servant aussi de capteur de température mais aussi de capteur de luminosité. Vient s'ajouter aux deux SensorTags, un anémomètre pour la capture de la vitesse du vent. Les SensorTags communiquent par Bluetooth, il a donc fallu utiliser un module BLE connectable directement sur la Waspmote.

L'anémomètre est directement branché sur un port analogique de la carte. Les données reçues sont ensuite envoyées par wifi sur un serveur distant sur lequel un utilisateur peut se connecter et voir les données sous la forme de graphiques. L'utilisateur peut aussi choisir de mettre le système en mode manuel et contrôler l'ouverture des fenêtres et des stores directement depuis une interface web.

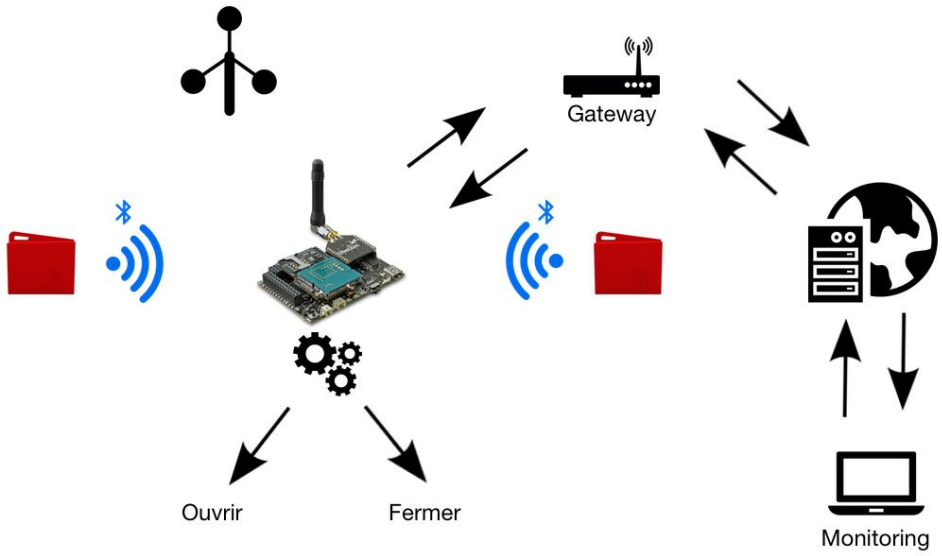

Figure 1: Schéma de la situation

Waspmote

Communication avec les Capteurs

Pour récupérer les données reçues par les SensorTags, nous avons d'abord utilisé le code d'exemple de l'éditeur Waspmote de connexion Bluetooth par le module BLE. Nous avons juste eu à retrouver les adresses MAC des senseurs pour pouvoir se connecter dessus. Nous avons ensuite utilisé un autre code d'exemple pour récupérer les données par notifications. Cette méthode permet de récupérer une donnée par seconde et non en continu pour éviter les erreurs.

Les données reçues étaient en hexadécimal mais un code de conversion est disponible sur l'API de Waspmote. Toutes ces fonctions mises ensemble nous ont permis de lire les valeurs de température de luminosité envoyées par les senseurs. De plus nous avons décidé de faire la moyenne des valeurs lues par la Waspmote afin de n'envoyer qu'une seule valeur au serveur.

L'anémomètre étant directement connecté à la carte nous avons juste eu à lire le signal analogique sur le bon pin. Nous avons quand même eu à convertir cette donnée car c'est une valeur de tension. Une simple règle de 3 permet de convertir la valeur de tension en résultat exploitable (m/s).

Envoi des données par WiFi

Pour la communication par WiFi entre la Waspmote et le serveur, nous avons été obligés d'utiliser un module WiFly. Ce module communique par UART avec la carte. Nous avons donc eu à configurer le WiFly sur TeraTerm et ouvrir les ports UART de la carte. Un code permet donc d'écrire sur les ports UART de la carte puis le WiFly vient les lire et les envois en TCP/IP sur le réseau. Un serveur connecté sur le réseau peut ensuite venir lire les données envoyées par le WiFly (dans notre cas le serveur est un Raspberry Pi). L'host du WiFly est le serveur.

Interprétation des résultats

Les données étant lues et envoyées, nous avons eu à penser à un algorithme d'ouverture automatique des fenêtres et des stores. Une simple comparaison entre la température intérieure et extérieure a finalement suffi. Si la température intérieure est supérieure à la température extérieure mais aussi supérieure ou égale à 25°C, la fenêtre s'ouvre. De plus, nous avons rajouté la condition du vent. S'il y a du vent, la fenêtre s'ouvre, même si la température extérieure est plus élevée.

Pour l'ouverture des stores nous avons fixé un seuil d'intensité lumineuse. Si l'intensité lumineuse est trop basse comparée à ce seuil (plus de 10 lux de différence), les stores s'ouvrent jusqu'à ce que le seuil soit atteint. Si le seuil n'est jamais atteint, les stores s'ouvrent complètement. La logique est inversée si l'intensité lumineuse est trop élevée. Nous avons-nous même fixé ce seuil à 150 lux en utilisant les résultats recueillis et en estimant quelle intensité lumineuse est agréable dans une pièce. Nous voulions faire en sorte que l'utilisateur puisse choisir ce seuil sur l'interface web mais nous ne l'avons pas fait faute de temps.

Schéma électrique

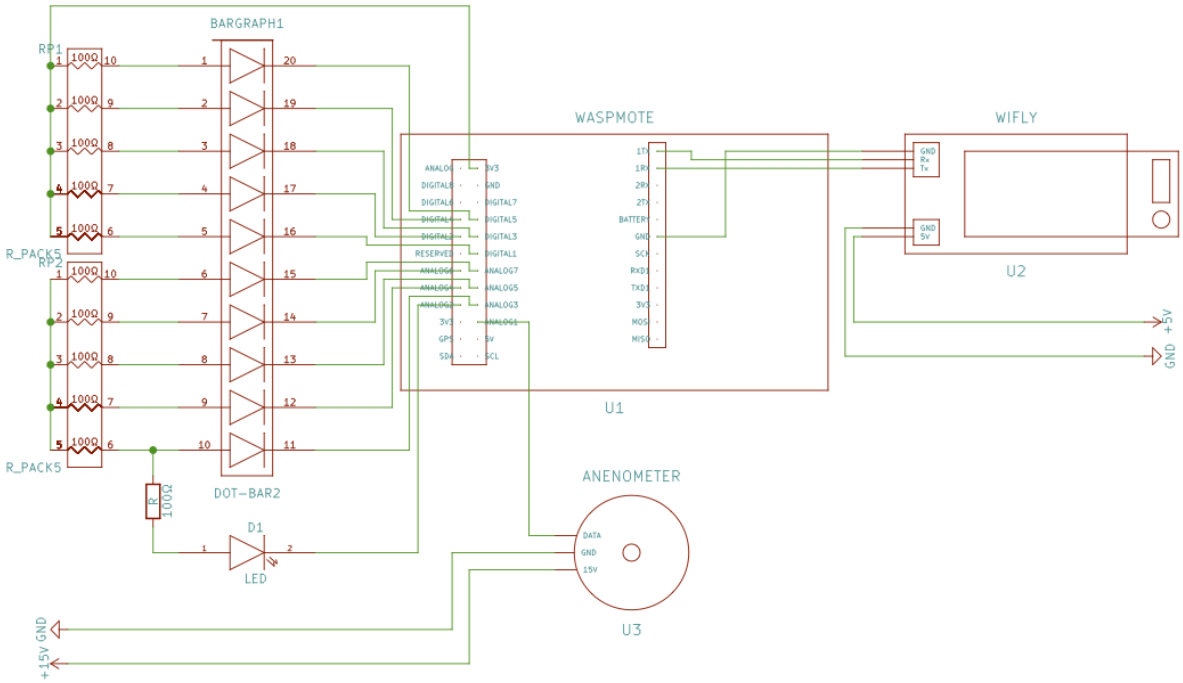

Figure 2: Schéma électrique

Serveur

Matériel

Notre serveur est un Raspberry Pi 3 avec Raspbian (une variante de Debian) installée dessus. Plusieurs étapes s'exécutent sur le serveur : réception des données provenant du WiFly, enregistrement de ces données en base de données, serveur Apache pour notre interface web et envoi de données en retour au WiFly.

Code Python

Nous avons utilisé python pour ouvrir un socket communicant avec la Waspmote et une communication avec la base de données. Pour le socket de la Waspmote, nous avions initialement essayé en PHP, mais le socket ne recevait rien (sûrement un mauvais code de notre part).

Le code python se divise en 3 threads (car nous devons en même temps recevoir les données de la Waspmote, les envoyer et les enregistrer en base de données) : un thread principal enregistrant la moyenne des valeurs de température, lumière et vent toutes les 10 minutes, un autre thread ouvrant un socket écoutant la Waspmote et un dernier lié à la base de données.

Le thread avec le socket qui écoute la Waspmote fonctionne ainsi : si le mode automatique est actif et qu'un changement d'état (fenêtre, store) est détecté, il enregistre en base de données les changements à la date et heure donnée. Le dernier thread sonde la base de données : si l'état inscrit en base diffère de l'état actuel (envoyé depuis la Waspmote), cela veut dire qu'il y a eu action de l'utilisateur et qu'il faut transmettre les changements. Pour se partager les données entre le thread principal et celui écoutant la Waspmote nous avons utilisé une Queue.

Base de données

Nous avons utilisé une base de données MySQL pour stocker nos données. Nous avons utilisé MySQL Workbench pour « dessiner » nos schémas et générer le code SQL. Nous possédons trois tables : une table « data », qui stocke les valeurs de température, luminosité, vent, date et heure, une autre « state » qui stocke l'état actuel du système (mode auto/manuel, ouverture/fermeture des fenêtres et stores) ainsi que l'utilisateur ayant déclenché l'action. La dernière table « users » stocke les utilisateurs avec leur nom, mot de passe et rôle (admin/user). La table « state » est liée à la table « users ».

Code Web

Notre code PHP est structuré en plusieurs fichiers : dans *database_connection.php* se trouvent toutes les fonctions permettant l'enregistrement des données et leur récupération, par jour ou par mois donnés, deux fonctions faisant de même pour l'état du système ainsi que des fonctions de récupération et ajout d'utilisateurs. Une ébauche d'espace d'administration a aussi été ajoutée, dans les fichiers *admin.php* et ses pages inclues et ses liens.

Dans *index.php* et *graphics.php* sont gérés l'affichage des données sous forme de tableau brut et de graphiques. Les graphiques ont étés réalisés avec Chart.js, un plug-in JavaScript bien pratique et le tri et recherche dans le tableau avec un autre plug-in, un fork de jQuery tablesorter. Le contrôle manuel du système est géré par un simple formulaire. Le design en CSS est basé sur Bootstrap CSS.

Nous avions commencé à étudier CakePHP, un framework PHP, mais nous avons abandonné l'idée, faute de temps et du fait que nous n'avions besoin que de quelques pages.