

AGINOV

# Wifi Gateway using Embedded AccessPoint

---

Responsible : Dr. Stephan Robert

**Jonathan Despraz**

**01/08/2013**

The purpose of this project is to implement a light embedded wibox into an AccessPoint device. The Wibox is a gateway between Wifi sensor and the server where the data will be display for final user. The Wibox was fully implemented in Java and run on Linux or Windows machine. Since the AccessPoint does not have much resources, all Wibox function won't be implemented.



## 1 Contenu

|       |   |    |
|-------|---|----|
| 1     | Introduction .....  | 4  |
| 1.1   | Cahier des charges : .....                                    | 4  |
| 1.2   | Présentation de la Wibox .....                                | 5  |
| 1.3   | Présentation des capteurs .....                               | 9  |
| 1.4   | Explication du fonctionnement de la Wibox .....               | 9  |
| 1.5   | Heartbeats, configuration et transmission de données .....    | 12 |
| 1.5.1 | Heartbeats : .....  | 12 |
| 1.5.2 | Linkup : .....  | 12 |
| 1.5.3 | Configuration : .....   | 12 |
| 1.5.4 | Data : .....  | 12 |
| 1.6   | Mémoire interne des capteurs .....                            | 13 |
| 1.7   | Présentation du Portal .....                                  | 13 |
| 1.8   | Installations des différents logiciels Wibox et Portal .....  | 14 |
| 1.8.1 | Installation de la base de données .....                      | 15 |
| 1.8.2 | Configuration des bases de données .....                      | 15 |
| 2     | Réalisation : .....   | 16 |
| 2.1   | Environnement de développement : .....                        | 16 |
| 2.2   | Environnement de programmation : .....                        | 17 |
| 2.2.1 | Notepad++ : .....   | 17 |
| 2.2.2 | CVS : .....   | 18 |
| 2.2.3 | Valgrind : .....  | 19 |
| 2.3   | Access Point : .....  | 19 |
| 2.3.1 | Configuration du point d'accès : .....                        | 20 |
| 2.4   | Analyse des paquets UDP envoyés par les capteurs : .....      | 21 |
| 2.5   | Protocole SNMP : .....  | 24 |
| 2.6   | Communication SNMP avec le capteur : .....                    | 26 |
| 2.7   | Acquittement d'un paquet de donnée auprès d'un capteur: ..... | 27 |
| 2.8   | Récupération de trappes SNMP : .....                          | 29 |
| 2.8.1 | Changement dans la réception des trappes SNMP : .....         | 31 |
| 2.9   | Récupérations des paquets « Heartbeat » : .....               | 31 |
| 2.10  | Reconfiguration des paramètres du capteur : .....             | 33 |
| 2.11  | Contrôle de saisie : .....                                    | 39 |

|        |   |    |
|--------|---|----|
| 2.12   | Présentation de SQLite .....                                      | 40 |
| 2.12.1 | Création d'une base de données test : .....                       | 41 |
| 2.13   | Création de la base de données réelle .....                       | 42 |
| 2.13.1 | Sensor_data : .....   | 43 |
| 2.13.2 | Sensor_information : .....  | 44 |
| 2.14   | Nettoyage de la base de données : .....                           | 45 |
| 2.15   | Lecture de la base de données : .....                             | 45 |
| 3      | Communication avec le Portal.....                                 | 49 |
| 3.1    | Types de paquets envoyés au Portal : .....                        | 49 |
| 3.2    | Implémentation de la communication avec le Portal : .....         | 51 |
| 3.3    | Présentation de RawCap : .....                                    | 51 |
| 3.4    | Analyse et envois des paquets à l'aide d'un serveur http : .....  | 52 |
| 3.5    | Implémentation du serveur http : .....                            | 52 |
| 3.6    | Connexion de la Wibox Lite au Portal : .....                      | 53 |
| 3.7    | Disposition des capteurs sur le réseau: .....                     | 54 |
| 3.8    | Découpage du programme en différents threads : .....              | 56 |
| 3.8.1  | Data_listener : .....   | 57 |
| 3.8.2  | Trap_listener : .....   | 57 |
| 3.8.3  | Command_listener : .....  | 57 |
| 3.8.4  | Snmp_listener : .....   | 58 |
| 3.8.5  | Server_http : .....   | 58 |
| 3.9    | Envoi des données au Portal : .....                               | 58 |
| 3.10   | Consommation mémoire et CPU du programme de la Wibox Lite : ..... | 60 |
| 3.11   | Correction de bugs et amélioration du programme : .....           | 60 |
| 3.12   | Plateforme embarquée Raspberry Pi .....                           | 62 |
| 3.13   | Démarrage du Raspberry Pi : .....                                 | 63 |
| 3.14   | Configuration du réseau sans fil sur le Raspberry Pi : .....      | 65 |
| 3.15   | Lancement du programme au démarrage du Raspberry Pi : .....       | 65 |
| 3.16   | Installation de la toolchain pour le Raspberry Pi : .....         | 66 |
| 4      | Mise en place de la Wibox Lite : .....                            | 71 |
| 4.1    | Reconfiguration des capteurs depuis un pc : .....                 | 74 |
| 5      | Conclusion : .....  | 76 |
| 6      | Bibliographie .....   | 77 |



|     |                          |    |
|-----|--------------------------|----|
| 7   | Annexes.....             | 79 |
| 7.1 | Journal de travail ..... | 79 |

## 1 Introduction

### 1.1 Cahier des charges :

L'objectif de ce projet est d'implémenter une version light et embarquée de la Wibox sur un AccessPoint. La Wibox est une passerelle entre les capteurs Wifi et un server où les données sont affichées à l'utilisateur. La Wibox est complètement implémentée en Java et fonctionne sous Linux ou sur une machine Windows. Comme un AccessPoint n'a pas énormément de ressources, la Wibox ne va pas être implémentée en intégralité. Le code Java de la Wibox est disponible pour l'étudiant.

Pour ce projet, un AccessPoint Netis OpenWrt sera utilisé. Par contre, le système devra être compatible (ou sans grands changements) avec d'autres périphériques embarqués fonctionnant sous Linux. Le langage de programmation ainsi que la base de données ne sont pas encore définis.

Voici la liste de priorité des fonctionnalités à implémenter:

- 1) Lire et interpréter les différents paquets en provenance des capteurs
- 2) Commencer à stocker les données du capteur dans une base de donnée light (la rétention des données devra être d'une ou deux semaines à cause du manque de mémoire flash). Les données doivent être acquittées auprès du capteur lorsqu'elles seront stockées dans la base de données locale
- 3) Afficher le(s) capteur(s) sur la page de configuration de l'AccessPoint dans un nouvel onglet appelé Aginova Sensor
- 4) Implémenter une reconfiguration basique des capteurs telle que la redéfinition des périodes et des réglages Wifi (fonctions restantes à définir par Aginova)
- 5) Implémenter l'API Wibox (Ce sera une version light puisqu'il n'y aura pas toutes les fonctionnalités). Cela se compose des protocoles XML-HTTP et XML-RPC
- 6) Afficher les données basiques à propos de l'état des capteurs (comme un graph de l'état de la batterie)
- 7) Rendre le système compatible avec une carte USB 3G pour le remote monitoring

## 1.2 Présentation de la Wibox

La Wibox est une passerelle entre les capteurs et un serveur où les données mesurées par les capteurs seront affichées à l'utilisateur final. La version complète de la Wibox a déjà été implémentée en langage Java par Aginova. La Wibox permet d'effectuer plusieurs actions comme la reconfiguration de certains paramètres propres aux capteurs et l'acquittement des paquets de données. Elle gère également toute la partie de communication « bas niveau » avec les capteurs.

Ce chapitre décrit les différentes fonctionnalités fournies par la Wibox. Ce travail de bachelor consiste à réaliser une version lite de la Wibox tournant sur un Linux embarqué, par conséquent, cette version n'intégrera pas toutes les fonctionnalités de la Wibox originelle.

Comme la Wibox est installée en local sur mon pc, l'accès se fait au travers d'un navigateur web en tapant l'adresse « <http://localhost:8088/> » dans la barre d'adresse. Nous arrivons ainsi directement sur la page d' « Overview » de celle-ci.

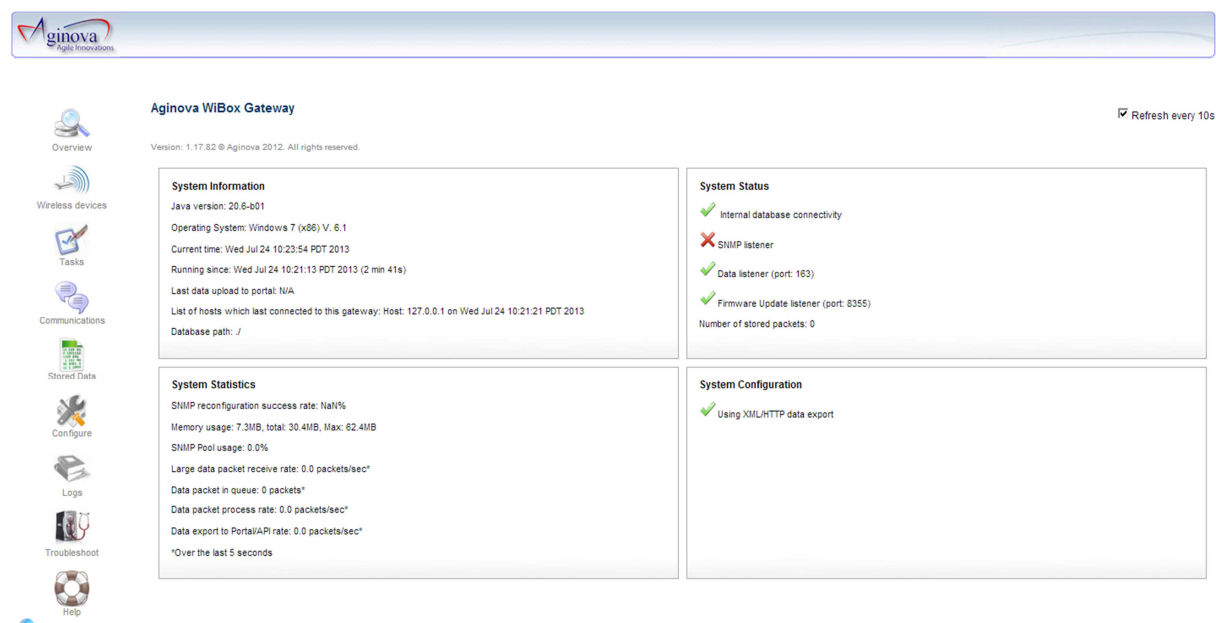


Figure 1 Aperçu de la Wibox

Sur la figure ci-dessus, nous pouvons voir la page d'accueil de la Wibox. Celle-ci est divisée en quatre parties. La partie en haut à gauche nommée « System Information » affiche des informations sur la plateforme sous laquelle tourne la Wibox comme la version de Java installée, le système d'exploitation l'heure actuelle etc... La seconde fenêtre nommée « System Status » informe l'utilisateur de l'état de la base de données et des écoutes sur les différents ports (Data, SNMP et Firmware Update) ainsi que sur le nombre de paquets stockés. La fenêtre suivante « System Statistics » affiche des informations relatives à l'utilisation de la mémoire, le pourcentage des paquets de données reçus etc... La dernière fenêtre nommée « System Configuration » nous informe, dans ce cas, que le système est configuré pour exporter les données selon le protocole XML/HTTP qui sera expliqué dans la suite du rapport.

Les différentes fonctionnalités offertes par la Wibox se trouvent sur la gauche de la fenêtre. Nous allons tout d'abord nous intéresser au menu « Wireless devices ». Ce menu affiche différentes

informations sur les capteurs connectés à la Wibox. On peut, par exemple, connaître l'ID du capteur, son adresse MAC, son voltage, la force de son signal ainsi que la valeur de plusieurs de ses paramètres comme les périodes de Linkup, Config et Sampling. Il est possible, à partir de ce menu, de reconfigurer n'importe quel capteur en le sélectionnant. Nous arrivons alors à une page de reconfiguration :

**Reconfigure sensor(s): 1**

This page lets you reconfigure sensors characteristics. Please leave the fields empty when you don't want to modify them.  
Current time: Sat Jun 01 12:35:19 CEST 2013

**Sensor Parameters**

|                         |  |                          |  |
|-------------------------|--|--------------------------|--|
| New Sensor ID:          | <input type="text"/>   | MAC Address:             | <input type="text"/><br><small>(ex 00:15:00:12:99:66)</small>              |
| Sampling period: ?      | <input type="text"/><br><small>(In sec, for example 2)</small> | Heartbeat period: ?      | <input type="text"/><br><small>(In sec)</small>                            |
| Configuration period: ? | <input type="text"/><br><small>(In sec, &gt;4 sec)</small>     | Sampling sending rate: ? | <input type="text"/><br><small>(number of sampling before sending)</small> |
| LinkUp period: ?        | <input type="text"/><br><small>(In sec)</small>                | Store & Forward: ?       | <input type="text" value="Don't change"/>                                  |
| Product:                | <input type="text" value="Don't change"/>                      |                          |  |

▶ WiBox Parameters  
 ▶ WiFi Parameters & Security  
 ▶ TCP/IP Parameters  
 ▶ Other Parameters  
 ▶ Calibration  
 ▶ Ultra Low Power Alarm Settings

(\*) Changing any of these settings might affect your connectivity with the WiBox/Desktop Software. Depending on your change, you will need to update your Access Poi

**Reconfigure**

Figure 2 Page de reconfiguration d'un capteur

Nous constatons qu'il est possible de reconfigurer un grand nombre de paramètres propres aux capteurs comme :

- L'ID du capteur
- Sa MAC Address
- Sampling period c'est-à-dire la fréquence à laquelle les paquets de données contenant les mesures vont être envoyés à la Wibox
- Heartbeat period, ce paramètre définit la fréquence à laquelle le capteur va envoyer des paquets contenant des informations sur l'état du capteur.
- Configuration period, ce paramètre définit la fréquence à laquelle le capteur va envoyer des paquets de configuration update qui vont permettre la reconfiguration du capteur.
- Sampling sending rate, c'est le nombre de paquets stockés dans la mémoire du capteur avant leur envoi.
- Linkup period, définit la fréquence à laquelle les paquets servant à maintenir la connexion wifi vont être envoyés au point d'accès.

- Store & Forward permet de définir si les paquets doivent être stockés sur le capteur si le réseau n'est pas disponible.
- Product, permet de définir le type de capteur utilisé.

Nous constatons qu'il y a encore de multiples onglets qui permettent notamment de reconfigurer les paramètres réseau du capteur. Les paramètres présent dans l'onglet « Wibox Parameters » permettent de changer l'adresse IP de la Wibox c'est-à-dire l'adresse à laquelle les capteurs vont envoyer leurs paquets de données. Il est aussi possible de changer le port UDP sur lequel les paquets de données sont envoyés ou encore le port utilisé pour la communication SNMP avec le capteur. Par défaut, le port de destination des données est le port 163 et le port SNMP est le port 161. Les paramètres DNS sont également reconfigurables sous l'onglet « Wibox Parameters ».

L'onglet « Wifi Parameters & Security » permet de configurer le réseau Wifi auquel le capteur va se connecter. Dans cet onglet, il est possible de configurer trois réseaux Wifi auquel le capteur va tenter de se connecter selon un ordre de préférence. Le premier réseau sera choisi si celui-ci est disponible sinon ce sera le second choix et si celui-ci n'est toujours pas disponible, ce sera le troisième réseau qui sera choisi. Pour chaque réseau Wifi, les paramètres suivant sont configurables :

- SSID, c'est-à-dire le nom du réseau sans fil auquel le capteur va se connecter.
- Channel, définit le canal du réseau Wifi à utiliser
- WPA 1/2 PSK, c'est le mot de passe servant à accéder au réseau Wifi désiré.
- WEP key, ceci est la clef de sécurité dans le cas de l'utilisation d'un réseau sécurisé par WEP.










Dans l'onglet « TCP/IP Parameters », il est possible de choisir d'utiliser le protocole DHCP (Dynamic Host Configuration Protocol) qui permet d'allouer dynamiquement des adresses IP aux différents appareils connectés au réseau Wifi ou alors de désactiver l'utilisation du protocole DHCP et d'entrer manuellement l'adresse IP du capteur à reconfigurer ainsi que son masque de sous réseau et l'adresse de sa passerelle réseau.

Les onglets de reconfigurations restants me paraissent moins intéressants et ne nécessitent pas d'être détaillés dans ce rapport car je n'aurais pas à implémenter leurs fonctionnalités dans la version Lite de la Wibox.

Outre la reconfiguration des différents capteurs, plusieurs paramètres peuvent être explorés en naviguant dans les différentes pages de la Wibox. Pour changer de page, il faut utiliser les boutons présents sur la gauche de la page de la Wibox. Nous nous sommes déjà intéressés à la page « Overview » qui affiche différentes informations sur le fonctionnement du système et à la page « Wireless devices » sur laquelle sont affichés les différents capteurs connectés à la Wibox et qui permet la reconfiguration de ceux-ci.

Une autre page extrêmement intéressante est la page nommée « Communications ». Cette page permet d'afficher les différentes informations envoyées par les capteurs connectés à la Wibox. Elle permet de sélectionner le capteur dont on veut afficher les informations, on peut également choisir d'afficher les paquets de données envoyés par le capteur. Les informations affichées sur cette page sont très utiles car elles permettent d'avoir un aperçu des différents paquets émis par le capteur

sélectionné. Il est possible de consulter les paquets de « Configuration Update », « Linkup », « Heartbeat », etc...

-  Overview
-  Wireless devices
-  Tasks
-  Communications
-  Stored Data
-  Configure
-  Logs
-  Troubleshoot
-  Help

### Communications

A list of the most recent packets exchanged between the WiBox and the Wifi Sensors.

Current time: Sat Jun 01 14:15:25 CEST 2013

Show only sensor:  , Show data packets:

| Time                   | Sensor ID | Direction      | Message      | Status | Description   |
|------------------------|-----------|----------------|--------------|--------|---|
| 6/1/2013 13:44:19 CEST | 1         | WiBox → Sensor | SNMP Set     | ✓      | set ACK to 1107 (new), End config for sensor (sent by WiBox to terminate the GET/SET window)  |
| 6/1/2013 13:44:19 CEST | 1         | Sensor → WiBox | ConfigUpdate | ✓      | (sent by the sensor to initiate a SET/GET from the WiBox)   |
| 6/1/2013 13:44:15 CEST | 1         | Sensor → WiBox | Heartbeat    | ✓      | APChannel=1, APMAC=00:23:69:41:75:80, assPrd=179, codeVersion=5.9.9, comMode=0, count=0, gparent=0, product=XTEMP-1015-0001, resets=0, retryCount=0.000, rssi=-39, storeFwdEnabled=1, tin |
| 6/1/2013 13:44:01 CEST | 1         | Sensor → WiBox | Heartbeat    | ✓      | APChannel=1, APMAC=00:23:69:41:75:80, assPrd=174, codeVersion=5.9.9, comMode=0, count=0, gparent=0, product=XTEMP-1015-0001, resets=0, retryCount=0.000, rssi=-38, storeFwdEnabled=1, tin |
| 6/1/2013 13:43:59 CEST | 1         | WiBox → Sensor | SNMP Set     | ✓      | set ACK to 1103 (new), End config for sensor (sent by WiBox to terminate the GET/SET window)  |
| 6/1/2013 13:43:59 CEST | 1         | Sensor → WiBox | LinkUp       | ✓      | (sent by the sensor to show that it is still alive to the AP)   |
| 6/1/2013 13:43:59 CEST | 1         | Sensor → WiBox | ConfigUpdate | ✓      | (sent by the sensor to initiate a SET/GET from the WiBox)   |
| 6/1/2013 13:43:47 CEST | 1         | Sensor → WiBox | Heartbeat    | ✓      | APChannel=1, APMAC=00:23:69:41:75:80, assPrd=170, codeVersion=5.9.9, comMode=0, count=0, gparent=0, product=XTEMP-1015-0001, resets=0, retryCount=0.000, rssi=-39, storeFwdEnabled=1, tin |
| 6/1/2013 13:43:39 CEST | 1         | WiBox → Sensor | SNMP Set     | ✓      | set ACK to 1099 (new), End config for sensor (sent by WiBox to terminate the GET/SET window)  |

Figure 3 Page « Communications » de la Wibox.

La page suivante nommée « Stored Data » donne des informations sur les paquets de données envoyés par les différents capteurs et stockés dans la base de données de la Wibox. Cette page offre également la possibilité d'effacer la base de données contenant les mesures faites par les différents capteurs.

### 1.3 Présentation des capteurs

Les Capteurs Wifi Sentinel sont des périphériques capables de communiquer avec plusieurs types de sondes comme des sondes de température, d'humidité, ou encore de lumière. Ces capteurs utilisent une infrastructure réseau de type Wifi 802.11b/g afin d'envoyer et recevoir des données.

Les capteurs vont prendre des mesures à un intervalle de temps régulier et envoyer le résultat de ces mesures à un serveur local. Lorsque les capteurs ne prennent pas de mesures, ils entrent dans un mode « low power » afin de préserver leur batterie interne au lithium.

Voici les caractéristiques du capteur Sentinel PRO II que j'ai utilisé pour mon travail de Bachelor :



Figure 4 Capteur Sentinel PRO II

- Capteur de température avec capteur d'humidité en option
- Jusqu'à deux sondes de capture
- Mesure des températures possible entre -30°C et +85°C
- 512K de mémoire flash
- LED signalant les alarmes et utile pour le débogage
- Bouton poussoir pour le diagnostic
- Store & forward (~30'000 mesures possible)
- Sécurité : WPA2-PSK, WPA1-PSK, WEP
- Alertes : email et SMS
- Durée de vie de la batterie avec Li AA : jusqu'à 3 ans

### 1.4 Explication du fonctionnement de la Wibox

La figure ci-dessous présente l'utilisation traditionnelle de la Wibox. Dans la configuration ci-dessous, le logiciel de la Wibox est installé sur un PC connecté par Wifi à un point d'accès. Les différents capteurs sont eux aussi connectés au même point d'accès et envoient leurs informations au PC.

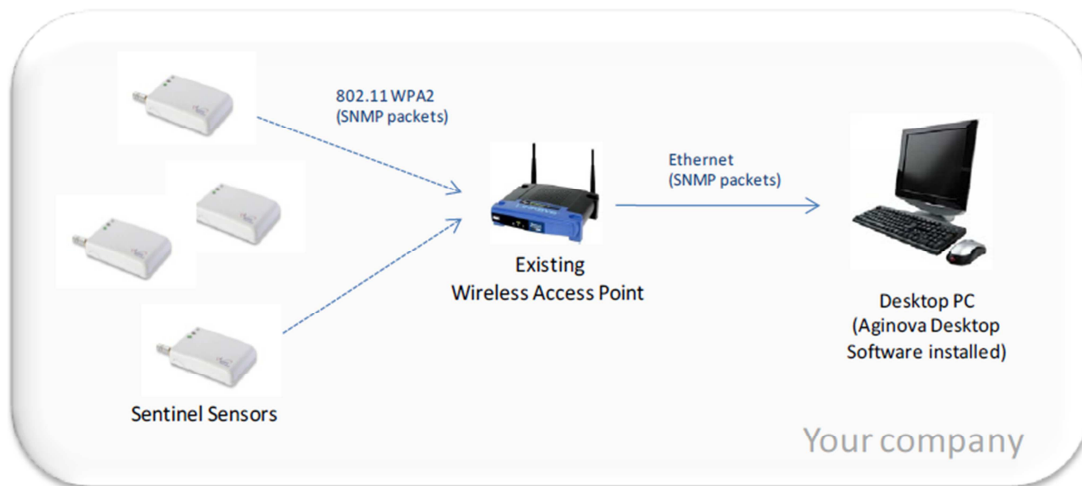


Figure 5

Les capteurs Sentinel sont configurés pour envoyer leurs données à l'adresse IP du PC sous lequel tourne le logiciel de la Wibox. Par défaut, cette adresse est 192.168.0.10 mais elle peut être configurée.

Les capteurs Sentinel vont d'abord essayer de se connecter au réseau Wifi en envoyant des paquets « probe requests » (IEEE 802.11) selon l'ordre suivant :

- SSID1, canal 1
- SSID2, canal 2
- SSID3, canal 3

Une fois que l'AccessPoint a répondu à ces requêtes, les capteurs vont effectuer les opérations suivantes :

- Authentification
- Association
- Setup security (EAP fast)
- Perform DHCP (seulement si le DHCP est activé)
- Effectuer les prochaines étapes

Les capteurs Sentinel vont effectuer plusieurs tâches basiques à des intervalles réguliers :

- Envoyer des messages « Heartbeat » (SNMP trap)
- Envoyer des messages « Configuration » (SNMP trap)
- Recevoir des messages « Configuration » (SNMP GET/SET)
- Envoyer des messages « Linkup » (SNMP trap)
- Mesures de températures (humidité pour SHT 11) et les stocker dans la mémoire flash
- Envoyer les mesures (messages UDP)

Lorsque le réseau Wifi n'est pas disponible, les capteurs vont effectuer des mesures séparées par un intervalle (sampling rate) et stocker les informations dans leur mémoire flash interne.



Lorsqu'un réseau Wifi compatible avec le capteur est disponible, le capteur va essayer de s'authentifier et de s'associer auprès du point d'accès. Une fois l'association complétée, il va envoyer des messages « Heartbeat » au PC sous lequel tourne la Wibox. Les messages « Heartbeat » vont permettre de transmettre l'ID du capteur (Sensor ID) à la Wibox ainsi que son adresse MAC et son voltage. Une fois que la Wibox a reçu au moins un message « Heartbeat » de la part d'un capteur, celui-ci est authentifié et la communication entre le capteur et la Wibox peut commencer.

Les capteurs Sentinel ont un algorithme d'envoi des données « low power » afin de préserver les batteries et d'effectuer des mesures de températures programmées, comme le montre la prochaine figure. Il n'est par contre pas possible pour la Wibox d'envoyer des messages de reconfiguration (ou des acquittements de données) sauf si le capteur est en phase d'attente de données.

Le capteur va se mettre mode attente des données à un intervalle nommé « configuration rate » et envoyé un message de configuration afin d'informer la Wibox qu'il est possible d'envoyer un message de reconfiguration au capteur.

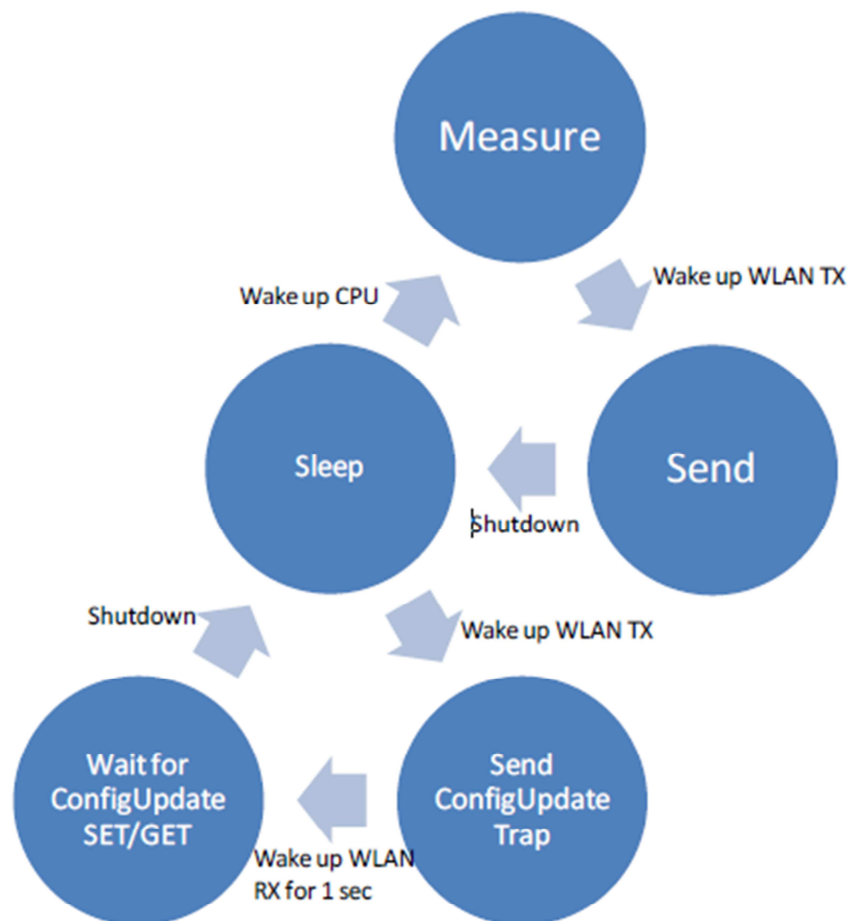


Figure 6 Fenêtre de configuration

## 1.5 Heartbeats, configuration et transmission de données

Comme expliqué précédemment, les capteurs Sentinel vont envoyer périodiquement des paquets de heartbeat, configuration, linkup ainsi que des données destinées à la Wibox. Ce chapitre va détailler les rôles de ces différents paquets.

### 1.5.1 Heartbeats :

Les heartbeats contiennent des informations sur l'ID du capteur, sa MAC adresse, l'adresse MAC du point d'accès connecté, une information temporelle, le voltage de la batterie, etc...

Les heartbeats sont encapsulés dans des paquets SNMP qui sont envoyés périodiquement au server. Ces paquets sont envoyés selon la méthode best effort, c'est-à-dire que le capteur ne va pas contrôler si les paquets ont été reçus correctement par le server. Les protocoles bas niveaux peuvent fournir un mécanisme de renvoi (par exemple, la couche MAC 802.11 va renvoyer deux fois le paquet avant d'abandonner).

Perdre des paquets contenant des informations de heartbeat n'est pas fatal, la seule incidence est que le server va croire que le capteur n'est plus connecté s'il ne reçoit plus de paquets de ce type pendant une certaine période de temps.

### 1.5.2 Linkup :

Les paquets de linkup sont des paquets SNMP envoyés au server afin de garder la connexion 802.11 ouverte. Quelques points d'accès vont perdre l'authentification des capteurs si aucun paquet n'est reçu pendant une certaine période (tous types de paquets confondus). Les paquets de linkup sont moins gourmands en termes de consommation d'énergie que les paquets de heartbeat.

Si l'intervalle entre les paquets de heartbeat est petit, alors ces paquets ne sont pas forcément nécessaires et peuvent être envoyés à une fréquence relativement faible. Tout comme les paquets de heartbeat, les paquets de linkup ne sont pas retransmis s'ils n'ont pas été reçus correctement par le server.

### 1.5.3 Configuration :

Le capteur va envoyer périodiquement des messages de reconfiguration au server. Lorsqu'un message de configuration est reçu par le server, celui-ci a une fenêtre d'une seconde afin de reconfigurer ou communiquer avec le capteur et refermer la fenêtre de reconfiguration.

Les messages de configuration sont encapsulés dans des paquets SNMP. Si le server ne referme pas la fenêtre de reconfiguration correctement, le capteur va essayer de renvoyer un message de configuration une fois de plus.

### 1.5.4 Data :

A un intervalle prédéfini, le capteur va effectuer une mesure et stocker le résultat de celle-ci dans sa mémoire flash interne. Si une connexion 802.11 est active avec le réseau, le capteur va essayer d'envoyer les données de la mesure au server.

Les données envoyées utilisent le protocole UDP dans un format propriétaire. Ces paquets peuvent être perdus lors de la transmission. Par conséquent, un algorithme d'acquittement est utilisé afin de garantir qu'aucune donnée ne puisse être perdue.

Toutes les données envoyées par le capteur au server sont indexées. Chaque fois que le capteur reçoit une donnée, il garde le dernier index continu dans sa mémoire. Lors du prochain message de reconfiguration avec le capteur, l'index va être envoyé de la Wibox au capteur. Grâce à ce processus, le capteur saura que les données ont été reçues correctement par le server. Si nécessaire, une donnée non acquittée sera à nouveau envoyée mais aucune donnée ne sera perdue.

## 1.6 Mémoire interne des capteurs

Les capteurs Sentinel vont stocker les données dans leur mémoire flash interne avant d'envoyer ces informations à la Wibox. Les capteurs, vont continuer d'essayer d'envoyer les données à la Wibox tant que celles-ci ne sont pas stockées proprement dans la base de données ou acquittées.

Ce mécanisme permet d'éviter que des données soient perdues en cas de faille temporaire du réseau ou lors de la communication sans fil. Afin de stocker proprement les données, les capteurs doivent avoir régler leur clock interne. Le clock est remis à zéro chaque fois que le capteur est éteint. Par conséquent, il est important de garder le capteur allumé lorsque celui-ci est utilisé en mode déconnecté.

## 1.7 Présentation du Portal

Le Portal est accessible par l'utilisateur final et permet de voir les données envoyées par les capteurs qui sont connectés à la Wibox. Sur la page d'accueil du Portal (onglet « Home »), on peut voir l'état des différents capteurs, si ceux-ci sont connectés ou non ou encore les données qu'ils ont envoyées. Il est également possible de sélectionner le type de capteur à afficher, c'est-à-dire les capteurs de lumière, de température ou d'humidité.

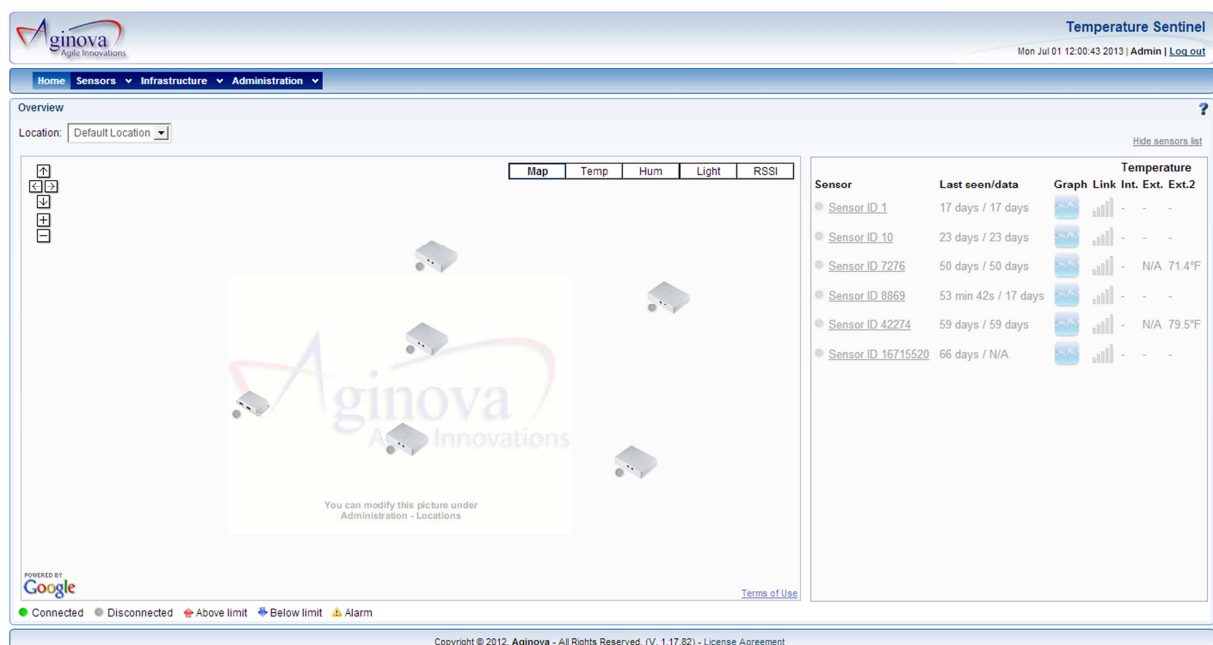


Figure 7 Onglet « Home » du Portal

Le Portal dispose de différents onglets, à côté de l'onglet « Home », se trouve l'onglet nommé « Sensors ». Cet onglet permet de sélectionner les options suivantes :

- Sensors Dashboard : Affiche l'état des capteurs, le nombre de capteurs actifs ainsi que des graphiques sur l'activité des capteurs.
- List of Sensors : Affiche la liste des capteurs utilisés ainsi que leur statut (connecté ou déconnecté) ainsi que la « Gateway » à laquelle ils sont connectés etc...
- Alarms : Permet de voir les alarmes présentes sur les capteurs.
- Graphs : Permet de sélectionner un capteur et de voir les données de celui-ci en fonction du temps sur un graphique.
- Data : Permet de voir les indications sur les données transmises par le capteur sélectionné.
- Report : Sert à imprimer les données d'un capteur sélectionné.
- Sensor signing : Affiche les informations de chaque capteur sous forme de graphiques

L'onglet suivant nommé « Infrastructure » permet de voir des statistiques sur la batterie restante des capteurs ainsi que sur les logs et les erreurs survenues lors de l'exécution du programme. Cet onglet me semble moins intéressant que les autres, c'est pourquoi je ne le détaille pas en profondeur dans ce rapport.

Pour finir, l'onglet « Administration » offre de nombreuses possibilités très intéressantes. Voici, ci-dessous, la description des sections qui me semble être les plus importantes dans cet onglet :

- Auditing : Permet de savoir quels paramètres ont été modifiés et par quel utilisateur ces changements ont été effectués.
- Gateways : Cette section est très importante, car elle permet de créer des « Gateway » comme la Wibox Lite par exemple. Pour créer une Gateway, il est nécessaire de savoir l'adresse IP de celle-ci ainsi que son port d'écoute.
- Setup : Cette section permet d'entrer les paramètres relatifs à la base de données à utiliser ainsi que le login et le mot de passe de celle-ci. D'autres paramètres de configuration nécessaires au système sont également présents sous cette section.

## 1.8 Installations des différents logiciels Wibox et Portal

La première étape nécessaire a été d'installer les logiciels Wibox et Portal. Le logiciel Wibox sert à recevoir les données des capteurs qui sont connectés directement sur l'AP prévu à cet effet ainsi qu'à reconfigurer les paramètres des capteurs, comme expliqué dans le début du rapport. Ce logiciel est très facile à installer sous Windows, il suffit juste de lancer l'exécutable d'installation qui va installer ce programme sans problème.

Pour installer le Portal Web, j'ai dû suivre une marche à suivre (Web Portal Installation Guide 1.17 rev 19.pdf) qui explique en détail les différents logiciels à installer. Ces logiciels sont les suivants : MySQL server Community et Apache.

Afin d'installer le Portal Web, la machine doit avoir les caractéristiques minimum suivantes :

- Windows 2000/XP Server or Red Hat Enterprise 4 AS/ES
- Administrator access
- Access to the sensors from/to ports 161-163
- Ability to bind ports 161-163, 8088, 80 (80 or other for web application)

- At least 1 GB of Physical RAM
- At least 1 GB of Hard Disk space
- MySQL Server 5.0+, PostgreSQL 8.0+, Oracle 10g R2 or MS SQL Server'05
- Java JDK 1.6
- Apache Tomcat 5.5 or BEA WebLogic Server 10.3
- Aginova WiBox Server

### 1.8.1 Installation de la base de données

Premièrement, il est nécessaire d'installer la base de données. Deux bases de données différentes ont été complètement testées pour le Portal Web, il s'agit de la base de données « PostgreSQL 8.0+ » (version 8.4.0-1 complètement testée). Cette base de données peut être téléchargée sur le site [www.postgresql.org](http://www.postgresql.org). L'installation de cette dernière est une installation basique c'est-à-dire qu'il est nécessaire d'installer toutes les fonctionnalités offertes par le programme et que l'on peut sans autre l'installer dans le dossier proposé par défaut.

La seconde base de donnée complètement testée pour le Web Portal est la base de données « MySQL 5.0+ » (version 5.5.8 complètement testée). Celle-ci peut être téléchargée à l'adresse suivante : <http://dev.mysql.com/downloads/mysql/>.

Si c'est la base de données MySQL qui a été choisie, il faut aussi installer MySQL Workbench qui peut être téléchargé à l'adresse suivante : <http://dev.mysql.com/downloads/workbench>. Le « Workbench » peut être installé complètement dans le dossier sélectionné par défaut.

### 1.8.2 Configuration des bases de données

Une fois la base de données correctement installée, il faut encore configurer cette dernière afin qu'elle puisse stocker correctement les données reçues. Les configurations complètes des bases de données « PostgreSQL 8.0+ » et « MySQL 5.0+ » sont également disponible dans le fichier pdf « Web Portal Installation Guide 1.17 rev 19.pdf ».

Une fois que celle-ci a été configurée correctement, il est nécessaire d'installer Apache Tomcat 5.5. Apache Tomcat est un logiciel open source qui implémente les spécifications des servlets et des JSP du Java Community Process. Il est paramétrable par des fichiers XML et inclut des outils pour la configuration et la gestion. Il comporte également un serveur HTTP. Apache Tomcat est téléchargeable à l'adresse suivante : <http://tomcat.apache.org>. La version testée de ce programme est la version 5.5.31. L'installation ainsi que la configuration de ce programme sont également détaillées dans le pdf « Web Portal Installation Guide 1.17 rev 19.pdf ».

## 2 Réalisation :

### 2.1 Environnement de développement :

J'ai réalisé ce projet sur une machine virtuelle tournant sous Linux. Comme la version finale de mon projet devra fonctionner sur une plateforme embarquée Linux, j'ai pensé qu'il serait judicieux de faire le développement logiciel directement sur Linux. Pour pouvoir utiliser une machine virtuelle, j'ai installé le logiciel VMware Workstation. Ce logiciel permet la création d'une ou plusieurs machines virtuelles au sein d'un même système d'exploitation. L'avantage de l'utilisation d'une machine virtuelle est que je peux avoir accès aux deux systèmes d'exploitation en même temps (Windows et Linux).

VMware est un très bon logiciel qui est complet et qui offre de nombreuses possibilités dans l'édition et les réglages des machines virtuelles. Il est par exemple possible de configurer le réseau de nombreuses manières ou encore de brancher divers périphériques à la machine virtuelle.

La machine virtuelle que j'ai créée fonctionne sous la distribution Linux Ubuntu 11.04 « Natty Narwhal ». J'ai utilisé cette distribution car elle m'est familière, je l'ai utilisée tout au long de mes études de plus, elle est fiable et fonctionne sans aucun problème. Comme cette machine virtuelle tourne sur mon ordinateur en parallèle avec Windows 7, j'ai dû lui allouer des ressources CPU et mémoire. Tout d'abord voici les caractéristiques de mon ordinateur portable sur lequel j'ai développé la totalité du projet :

- Modèle: Lenovo Thinkpad T410
- Processeur : Intel Core i7 CPU M620 @ 2.67GHz 2.67 Ghz (Quadcore)
- Système d'exploitation : Windows 7 Professionnel 64 bits
- Disque dur : 250 GB
- Mémoire RAM : 4 GB

Ensuite, voici les ressources que j'ai allouées à ma machine virtuelle Linux :

- Mémoire : 2.5 GB
- Nombre de processeurs : 2
- Taille du disque dur : 20 GB
- Interface réseau : Bridged
- USB Controller : Present

Lors de la réalisation de ce projet, j'ai dû changer quelques fois le mode d'interfaçage du réseau pour des raisons que j'ai expliquées dans les chapitres concernés.



Figure 8 Aperçu de la machine virtuelle Linux

## 2.2 Environnement de programmation :

Afin de développer mon programme en langage C, j'ai utilisé l'éditeur de texte nommé Kate, qui est disponible pour les systèmes d'exploitation GNU/Linux, BSD et autres systèmes apparentés à Unix. J'ai découvert cet éditeur de texte, spécialement conçu pour la programmation, lors de mes études et je le trouve très intéressant. Il propose différentes fonctionnalités utiles et agréables comme :

- Coloration syntaxique pour une centaine de formats, pliage de code
- Édition de plusieurs fichiers en simultanée possible
- Intégration de l'émulateur de terminal Konsole dans l'interface, grâce à la technologie Kparts
- Capable d'éditer des fichiers de taille importante sans ralentissements majeurs
- Possibilité d'utiliser des plugins

### 2.2.1 Notepad++ :

J'ai aussi utilisé quelque peu le logiciel Notepad++ qui fonctionne sous Windows et qui est très agréable à utiliser. Ce logiciel permet aussi d'utiliser de nombreux plug-ins qui sont très utiles, comme un plugin permettant de ré indenter du code ou encore la coloration syntaxique du code.

J'utilise très souvent Notepad++ pour apporter de légères corrections au code sans le recompiler, car Notepad++ fonctionne sous Windows et non sous Linux, je n'ai donc pas la possibilité de compiler mon code. Par contre, Notepad++ est très léger et très bien conçu, c'est pourquoi je l'utilise fréquemment pour éditer mon code sans devoir démarrer ma machine virtuelle ce qui demande beaucoup de temps.



## 2.2.2 CVS :

CVS pour « Concurrent versions system » est un système de gestion des versions possédant une licence GNU GPL et largement utilisé dans le développement de nombreux projets. J'ai installé ce logiciel sous ma machine virtuelle linux afin de pouvoir sauvegarder et recharger facilement les différentes versions de mon code.

CVS utilise une architecture client-serveur, un serveur stocke les versions courantes d'un projet et leur historique. Typiquement, le client se connecte au serveur par LAN ou par internet mais le client et le serveur peuvent aussi tourner sous une même machine si CVS est utilisé seulement par un développeur. Le logiciel du serveur fonctionne normalement sous Unix, tandis que le logiciel client de CVS fonctionne sous la plupart des systèmes d'exploitation.

CVS est un logiciel très utile car, en plus de permettre des backups, il permet à plusieurs développeurs de travailler sur un même projet en possédant chacun leur propre copie du projet en mémoire. Afin d'éviter que les développeurs n'efface les versions de leurs collègues, le serveur n'accepte que les changements de la version la plus récente d'un fichier. Les développeurs sont donc appelés à garder leur copie de travail en intégrant les changements d'autres personnes sur une base régulière.

Comme ce projet de bachelor est développé pour la société Aginova, c'est dans les bureaux de la société que se trouve le serveur contenant mon projet sous CVS. Cette solution est très utile, car si j'ai une question sur mon code, il me suffit de le mettre à jour sous CVS, puis les ingénieurs travaillant chez Aginova pourront regarder mon code dans sa dernière version afin de répondre à ma question. Cela simplifie grandement la vie, car il n'est pas nécessaire d'envoyer des fichiers par mail ou d'autres moyens alternatifs pour trouver une solution.

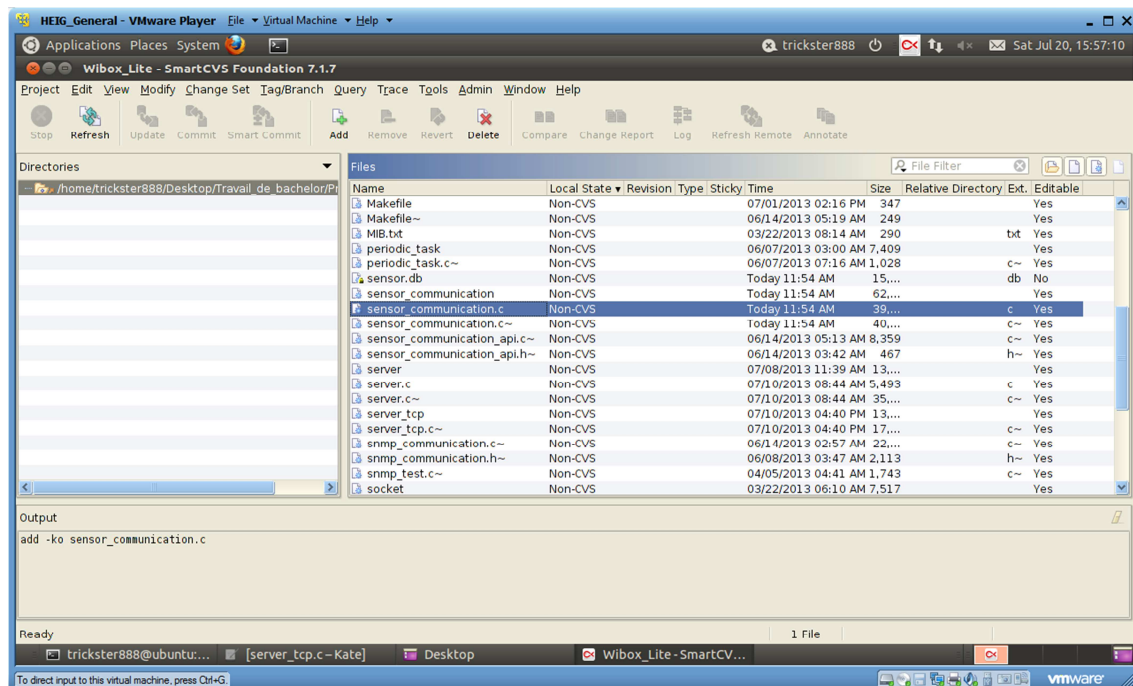


Figure 9 CVS



### 2.2.3 Valgrind :

Afin de me faciliter la vie lors du développement de mon programme, j'ai utilisé l'utilitaire Valgrind qui permet notamment de mettre en évidence des fuites mémoires. Cet outil est très utile et permet de gagner un temps précieux lors de l'élaboration d'un programme.

Valgrind est composé d'un cœur de différents outils mais celui que j'utilise le plus est l'outil par défaut qui est nommé « memcheck ». Memcheck analyse toutes les allocations, libérations et tous les accès mémoires. Utilisé Valgrind ralentit fortement le programme mais permet de récupérer des informations très utiles lors de bug ou de fuites mémoires.

Les principaux types de problèmes signalés sont les suivants :

- Accès hors d'une zone allouée (dépassement d'un tableau, accès à un indice négatif...)
- Libération d'une zone non allouée (pointeur corrompu) ou déjà libérée
- Utilisation d'une zone non initialisée
- Fuite mémoire

Pour lancer l'exécution du programme avec Valgrind, il suffit juste de rajouter le mot « valgrind » avant le nom du programme, dans mon cas j'ai juste à écrire « sudo valgring ./Wibox\_lite ». Il est aussi possible d'ajouter des options à Valgrind mais ces options sont à ajouter avant notre commande sinon elles seront transmises directement au programme.

Une fois que Valgrind est lancé, il va afficher un message indiquant qu'il charge memcheck puis il va commencer à afficher les erreurs au fur et à mesure qu'elles vont être rencontrées.

## 2.3 Access Point :

Pour pouvoir connecter les capteurs à mon ordinateur portable et à la plateforme Linux embarquée, j'ai utilisé un point d'accès Netis WF-2403.



Figure 10 Netis WF-2403

Ce point d'accès, qui m'a été fourni par Aginova, est un point d'accès portable qui ne dispose que d'un port Ethernet. Il est néanmoins possible de l'utiliser en trois modes différents grâce à un bouton situé sous celui-ci. Les différentes modes d'exécution possibles sont :

- Wireless Router
- Access Point
- Repeater
- Client
- 4-port Ethernet Switch

Voici les différentes caractéristiques de ce point d'accès :

- Standards : 802.11 b/g/n
- Fréquence : 2.412GHz – 2.4835GHz
- RF Power : 11n : 13 dBm, 11g : 14 dBm, 11b : 18 dBm
- Puissance : basse consommation, alimenté par un port USB
- Compatibilité : USB 2.0, USB 1.1
- OS supportés : Microsoft Windows 7, Vista, XP, 2000 SP4

### 2.3.1 Configuration du point d'accès :

Le point d'accès est configuré comme ci-dessous :

| Version           |  |
|-------------------|--|
| Hardware Version: | WF-2403  |
| Firmware Version: | APR-R4A4-V1.1.260EN-Netis(WF-2403),2011.05.26 15:50, |
| LAN               |  |
| MAC Address:      | 08:10:75:2b:39:68                                    |
| IP Address:       | 192.168.0.1  |
| Subnet Mask:      | 255.255.255.0  |
| DHCP Server:      | Enable   |
| Wireless          |  |
| Wireless Status:  | Enable   |
| Name(SSID):       | aginova  |
| Mode:             | Access Point   |
| Channel:          | 6  |
| MAC Address:      | 08:10:75:2b:39:68                                    |
| WPS Status:       | Enable   |
| Router Status     |  |
| System Uptime:    | 0 Days 0 hours 46 minutes 55 seconds                 |
| CPU Usage:        | 1%   |
| Memory Usage:     | 5%   |

Figure 11 Configuration du point d'accès Netis WF-2403

| Wireless Security   |  |
|---|--|
| For the security of your wireless network, we strongly recommend you to use the authentication type of WPA2-PSK, encryption type of AES or authentication type of WPA2-PSK, encryption type of TKIP&AES |  |
| Authentication Type:  | WPA2-PSK   |
| Encryption Type:  | <input type="radio"/> TKIP <input checked="" type="radio"/> AES <input type="radio"/> TKIP & AES |
| Key Mode:   | <input type="radio"/> HEX <input checked="" type="radio"/> ASCII                                 |
| Key:  | aginova1234 (please enter any 8-63 charcters (ASCII charcters:A-Z,a-z,0-9))                      |
| Key Renewal:  | 86400 seconds(60-86400)  |
| <input type="button" value="Save"/>   |  |

Figure 12 Configuration de la sécurité wifi du point d'accès

## 2.4 Analyse des paquets UDP envoyés par les capteurs :

Pour analyser les paquets UDP envoyés par les différents capteurs, j'ai d'abord utilisé le logiciel Wireshark sous Windows pour voir si les paquets arrivaient bien à destination. Pour que les paquets soient correctement envoyés, j'ai connecté le point d'accès contenant le SSID « aginova » à mon ordinateur grâce à un câble Ethernet, puis j'ai allumé un capteur et démarré la Wibox. Une fois que les paquets étaient visibles et correctement acheminés, j'ai configuré le réseau de ma machine virtuelle afin de recevoir et pouvoir traiter ces différents paquets sous Linux, car c'est sous Linux que je vais développer le programme de la Wibox Lite. Voici la configuration de la carte réseau de ma machine virtuelle :

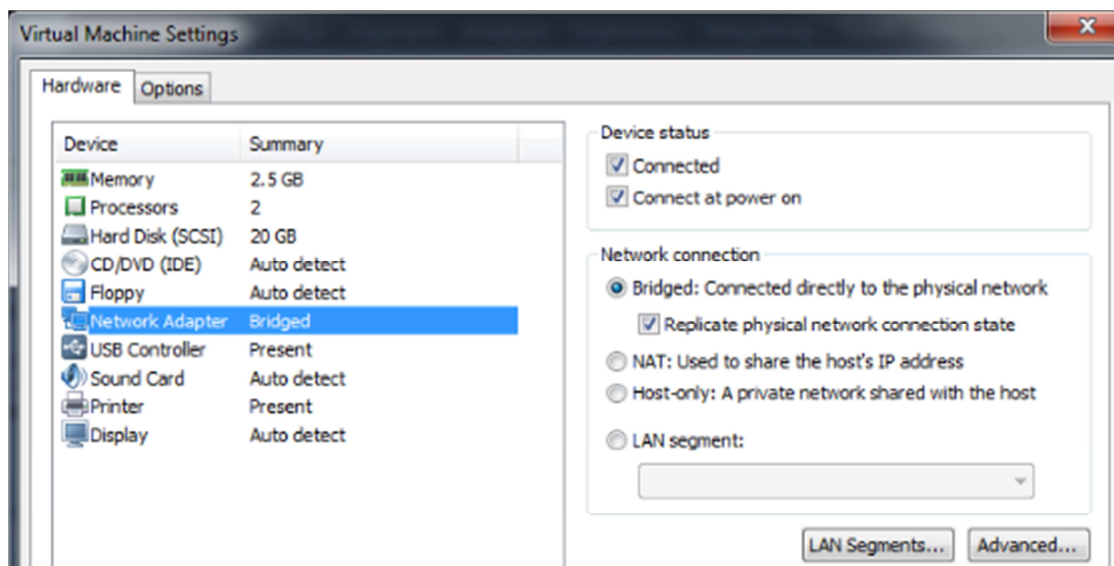


Figure 13 Configuration réseau de la machine virtuelle

J'ai ensuite créé un programme en langage C utilisant les sockets ce qui va me permettre d'analyser les paquets reçus. La communication se fait au travers du protocole UDP qui est un protocole qui n'est pas orienté connexion. Pour recevoir les paquets UDP, j'ai dû configurer l'interface eth0 de ma machine virtuelle avec l'adresse 172.19.0.10 et le masque de sous-réseau 255.255.255.0, car les paquets envoyés par les capteurs ont cette adresse comme adresse de destination. Il est également

nécessaire de désactiver le firewall de mon ordinateur afin de pouvoir laisser passer les paquets UDP sans filtrage. Pour pouvoir faire des tests avec le logiciel Wibox complet fonctionnant sous Windows ainsi qu'avec la machine virtuelle tournant sous Linux, je décide de changer l'adresse de destination des paquets et de mettre l'adresse de broadcast du sous réseau comme adresse de destination ce qui va bien me simplifier les choses pour faire différents tests. L'adresse de destination des capteurs est donc 172.19.0.255.

```
eth0      Link encap:Ethernet  HWaddr 00:0c:29:60:05:fe  
          inet addr:172.19.0.255 Bcast:172.19.0.255  Mask:255.255.255.0
```

Figure 14 Configuration de l'interface eth0

Voici les différents ports utilisés par l'application :

| Packet Name | Type | Source Port | Destination Port |
|-------------|------|-------------|------------------|
| Data        | UDP  | Random      | 163              |
| SNMP        | UDP  | 161         | 162              |
| FwUp        | UDP  | 80          | 8355             |

Figure 15 Ports utilisés pour l'envoi des paquets UDP

Pour commencer j'ai créé une application permettant de capturer les paquets SNMP arrivant sur le port 162. Cette application utilise les sockets UDP en C et permet de définir le port d'écoute afin de capturer les trames arrivant sur le port choisi et mettre les informations désirées dans un buffer pour pouvoir les traiter.

Pour pouvoir capturer des paquets UDP, grâce à la programmation socket, il est tout d'abord nécessaire d'inclure les bibliothèques destinées au socket sur l'environnement désiré. Comme je programme sous Linux, j'inclus les bibliothèques ci-dessous :

```
#include <sys/types.h>  
#include <sys/socket.h>  
#include <netinet/in.h>  
#include <arpa/inet.h>
```

Figure 16 Bibliothèques nécessaires pour la programmation socket

Il est maintenant possible de créer un socket à laquelle on va devoir spécifier une adresse d'écoute ainsi qu'un port d'écoute. Il faut aussi réserver un espace mémoire afin de mettre en mémoire les données récupérées par le socket.

```
// Port logiciel d'ecoute
int PORT = 162, n = 0, i = 0;

SOCKET sock = socket(AF_INET, SOCK_DGRAM, 0);
if(sock == INVALID_SOCKET)
{
    perror("socket()");
    return 0;
}

// Definition des parametres de la socket
SOCKADDR_IN sin = { 0 };
sin.sin_addr.s_addr = htonl(INADDR_ANY);
sin.sin_family = AF_INET;
sin.sin_port = htons(PORT);

if(bind (sock, (SOCKADDR *) &sin, sizeof sin) == SOCKET_ERROR)
{
    perror("bind()");
    return 0;
}
```

Figure 17 Déclaration et paramétrage d'un socket

Le protocole de communication détaillé entre les capteurs et l'AP est décrit dans le document « Sensor-communication-wiki.docx » ainsi que dans le document « WiBox API Developer Guide 1.17 rev 24.pdf » qui m'ont été fournis par Aginova.

J'analyse donc les paquets contenant les heartbeats. Ces paquets sont envoyés par chaque capteur à intervalle constant et contiennent des informations sur le capteur comme son ID, son voltage ou encore son adresse MAC.

Afin d'analyser ces différents types de paquets, je me contente de capturer les données brutes et d'analyser ensuite le buffer contenant ces données. Cette analyse permet de me favoriser avec le type de donnée transmise et le protocole de communication utilisé par les capteurs. J'ai également analysés les paquets arrivant sur le port 163 et contenant les données transmises par le capteur et communiquant à l'aide d'un protocole propriétaire.

Pour développer ce projet, j'ai utilisé un capteur de type Sentinel Pro 2 dont voici la description des paquets de données :

**[edit] Type 5****Usage:**

- Sentinel Pro 2 (with SHT11)

**Description:**

- Up to 16bit, temperature measurement (external probe 1) (ZoomingADC) (null=0x8000)
- Up to 16bit, temperature measurement (external probe 2) (ZoomingADC) (null=0x8000)
- Up to 16bit, temperature measurement (internal probe) (null=0xFFFF)
- Up to 16bit, humidity measurement (null=0xFFFF)
- Up to 16bit, light measurement (null=0xFFFF)
- Up to 16bit, NOT DEFINED YET

**Format:**

```
RAW_TEMP_EXT_ADC_1 reading: 2 bytes
RAW_TEMP_EXT_ADC_2 reading: 2 bytes
RAW_TEMP_INT_ADC reading: 2 bytes
RAW_HUM_ADC reading: 2 bytes
RAW_LIGHT_ADC reading: 2 bytes
NOT_DEFINED_YET: 2 bytes
```

Figure 18 Format des données pour le capteur Sentinel Pro 2

Comme les paquets arrivant sur le port 162 sont encapsulés dans le protocole SNMP, il est bien plus simple et plus propre de traiter ces paquets à l'aide d'un utilitaire open source comme expliqué ci-dessous.

## 2.5 Protocole SNMP :

Les capteurs communiquent avec l'AP à travers le réseau Wifi en utilisant le protocole SNMP (Simple Network Management Protocol). Ce protocole très complet permet de gérer différents équipements du réseau. Il comporte différentes versions utilisant des niveaux de sécurité différents. Dans ce projet, c'est la version 1 du protocole qui va être utilisée. Il faut savoir que les paquets de heartbeat, configuration et linkup sont encapsulés dans des paquets SNMP. L'architecture de ce protocole repose sur trois éléments principaux :

- Les équipements managés c'est-à-dire les capteurs
- Les agents, dans notre cas l'application réseau qui va gérer le management des capteurs
- Les systèmes de management réseau

Tous les équipements managés sont regroupés dans une base de données qui est organisée sous forme d'un arbre et qui peut-être contenu dans un fichier texte. Ces fichiers sont appelés les fichiers MIB pour Management Information Base. Il existe plusieurs utilitaires pour parcourir ou créer les fichiers MIB.

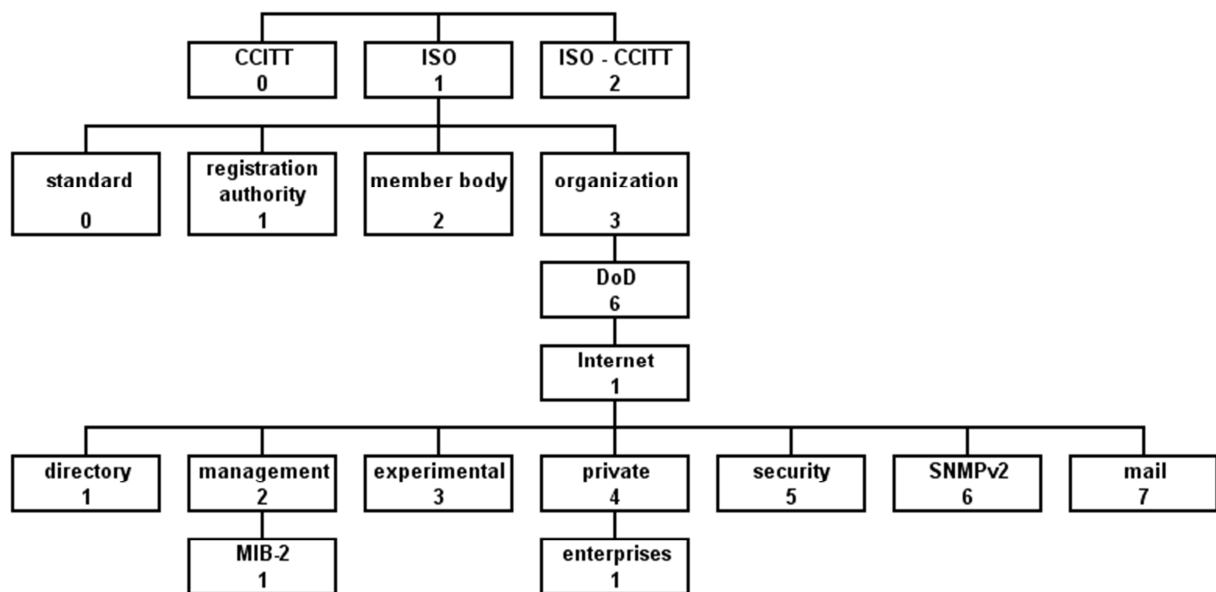


Figure 19 Représentation de l'arbre d'un fichier MIB

Afin de faire les choses proprement, j'ai téléchargé et installé sur ma machine virtuelle Linux l'utilitaire Net-SNMP qui est disponible en téléchargement libre à l'adresse suivante : <http://www.net-snmp.org/download.html>. Cet utilitaire open source met à disposition de nombreux outils afin de pouvoir recevoir des trappes SNMP, parcourir les fichiers MIB ainsi que des bibliothèques pour le langage C qui permettent de créer facilement des paquets de données au format SNMP. Ces bibliothèques me sont très utiles car je les utilise pour créer les requêtes SNMP afin de communiquer avec les capteurs.

Les différents capteurs ainsi que leurs fonctions sont donc désignés sous différents numéros nommés OID (Object Identifier) qui sont contenus dans les fichiers MIB. Karl m'a donc donné un fichier nommé Mote.java qui contient les OID des différents capteurs.

```

/**
 * New ACK system to avoid data corruption
 */
public static String AGINOVANEWACK = "1.3.6.1.4.1.28295.99.1.2.1.52.5";

/**
 * Sentinel Power reset energy counters
 */
public static String RESET_ENERGY_COUNTERS = "1.3.6.1.4.1.28295.99.1.2.1.61.5";

/**
 * Sentinel Power Zero Power detection
 */
public static String ZERO_POWER_DETECTION = "1.3.6.1.4.1.28295.99.1.2.1.60.5";

/**
 * EAP-FAST user name
 */
public static String GSNUSERNAME = "1.3.6.1.4.1.28295.1.1.2.8.0";

```

Figure 20 Extrait du fichier Mote.java



Grâce à ces différents OIDs, je vais savoir quelles valeurs je dois envoyer afin de modifier les différents paramètres du capteur ou quelle valeur envoyé afin d'acquitter les paquets de données envoyés à la Wibox lite. Afin de communiquer avec ces capteurs, j'ai créé un programme de communication avec les capteurs que j'ai nommé Wibox\_lite.c. La réalisation de ce programme est détaillée dans les chapitres suivants.

## 2.6 Communication SNMP avec le capteur :

Le programme de communication avec les capteurs doit bien sûr inclure les librairies Net-SNMP afin de pouvoir utiliser les différentes fonctions offertes par ces librairies, notamment la création de paquets SNMP. Ces librairies se nomment « net-snmp/net-snmp-config.h » et « net-snmp/net-snmp-includes.h ».

Pour initialiser le processus de communication du programme, il est nécessaire de créer une session SNMP qui va contenir différents paramètres propres à cette session. Voici la configuration des différents champs de cette session :

- version : Ce champ contient la version du protocole utilisée dans notre cas c'est la constante `SNMP_VERSION_1` qui lui est assignée car c'est la version 1 du protocole qui est utilisée.
- Community : Ce champ comporte le secret partagé entre les capteurs et l'agent SNMP c'est l'élément principal de la sécurité dans cette version du protocole cette valeur contient le string « `GSN_SET` »
- Peername : Ce champ contient l'adresse IP du capteur avec lequel on veut communiquer.
- Retries : Ce champ contient le nombre de requêtes que l'on souhaite envoyé au capteur si la première requête a échoué. J'ai, pour l'instant fixé ce nombre à 0.
- Timeout : Nombre en microsecondes avant le premier timeout. Pour l'instant j'ai fixé ce nombre à 10000000 c'est-à-dire 10 secondes.

Exemple de l'initialisation d'une session SNMP :

```
// Initialisation de la communication SNMP
init_snmp("Aginova");

// Creation de la session SNMP
snmp_sess_init(session);
session->version = SNMP_VERSION_1;
session->community = community;
session->community_len = strlen(session->community);
session->peername = *ip_address;
session->retries = 0;
session->timeout = 10000000;
snmp_synch_setup(session);
sess_handle = snmp_open(session);
```

Figure 21 Initialisation d'une session SNMP

La liste de tous les champs paramétrant une session SNMP est disponible à l'adresse suivante : [http://net-snmp.sourceforge.net/dev/agent/structsnmp\\_session.html](http://net-snmp.sourceforge.net/dev/agent/structsnmp_session.html).



Une fois que la session SNMP est ouverte, il est nécessaire de récupérer le fichier MIB contenant les différents OIDs. Cette opération peut se faire au travers des fonctions `add_mibdir(const char *)`, qui définit le chemin vers le dossier contenant le fichier MIB et la fonction `read_mib(const char *)` qui prend en paramètre le nom du fichier MIB à récupérer et qui retourne un pointeur vers une structure de type `tree`. Pour commencer, j'ai fait un fichier MIB qui contient les différents OID des capteurs et qui se nomme `Aginova.txt`.

```
// Recuperation du fichier MIB
add_mibdir(".");
mib_tree = read_mib("Aginova.txt");
```

Figure 22 Recupération du fichier MIB

J'ai utilisé la méthode nécessitant le fichier `Aginova.txt` pendant quelque temps afin de gérer les différents OIDs dont mon programme avait besoin. Je me suis ensuite rendu compte que l'élaboration du fichier `Aginova.txt` contenant l'arbre avec les différents OIDs devenait relativement compliquée, c'est pourquoi j'ai décidé d'utiliser la fonction nommée `snmp_parse_oid()`. Cette fonction, qui prend une chaîne de caractère ainsi qu'un objet de type `OID` en paramètre, va associer la chaîne de caractère à l'objet `OID` ce qui est beaucoup plus simple. J'ai alors déclaré des chaînes de caractère contenant tous les `OID` dont j'avais besoin au début de mon programme.

Une fois que la session a été initialisée, il faut créer la charge utile (PDU) qui contiendra les informations nécessaires au paquet SNMP. Il faut tout d'abord spécifier le type de la requête grâce à différentes constantes comme `SNMP_MSG_SET`, `SNMP_MSG_GET` ou `SNMP_MSG_RESPONSE`. Ces requêtes servent à interroger les variables d'activités de l'appareil sélectionné. Par exemple, pour l'acquittement des paquets, une variable contenant un entier représentant le numéro du paquet à acquitter va être envoyé depuis la Wibox au capteur désiré. Ainsi, grâce aux variables associées aux `OID` contenue dans les requêtes SNMP, il est possible de reconfigurer un capteur ainsi que d'acquitter ses paquets, de changer son adresse IP, son masque de sous-réseau, etc...

La reconfiguration des capteurs ainsi que l'acquittement des paquets de données grâce aux requêtes SNMP sont expliqués dans les chapitres suivant.

## 2.7 Acquittement d'un paquet de donnée auprès d'un capteur:

Voici une capture wireshark de l'acquittement d'un paquet fait par la Wibox :

|     |           |              |              |      |                  |
|-----|-----------|--------------|--------------|------|------------------|
| 130 | 35.442105 | 172.19.0.100 | 172.19.0.255 | SNMP | 119 trap iso.3.6 |
| 131 | 35.468603 | 172.19.0.12  | 172.19.0.100 | SNMP | 114 set-request  |
| 136 | 35.854331 | 172.19.0.100 | 172.19.0.12  | SNMP | 116 get-response |

Figure 23 Echange de messages SNMP entre la Wibox et un capteur

```

+ Frame 131: 114 bytes on wire (912 bits), 114 bytes captured (912 bits)
+ Ethernet II, Src: Wistron_0c:a8:32 (5c:ff:35:0c:a8:32), Dst: Gainspan_0a:0f:1d (00:1d:c9:0a:0f:1d)
+ Internet Protocol Version 4, Src: 172.19.0.12 (172.19.0.12), Dst: 172.19.0.100 (172.19.0.100)
+ User Datagram Protocol, Src Port: snmptrap (162), Dst Port: snmp (161)
+ Simple Network Management Protocol
  version: version-1 (0)
  community: GSN_SET
  data: set-request (3)
    set-request
      request-id: 229956589
      error-status: noError (0)
      error-index: 0
      variable-bindings: 2 items
        1.3.6.1.4.1.28295.99.1.2.1.52.5:
          object Name: 1.3.6.1.4.1.28295.99.1.2.1.52.5 (iso.3.6.1.4.1.28295.99.1.2.1.52.5)
          value (Integer32): 7767
        1.3.6.1.4.1.28295.1.1.4.6.0:
          object Name: 1.3.6.1.4.1.28295.1.1.4.6.0 (iso.3.6.1.4.1.28295.1.1.4.6.0)
          value (Integer32): 1

```

Figure 24 Détails du paquet contenant l'acquittement envoyé au capteur

Dans cette capture d'écran, on constate que le capteur envoie une trappe SNMP qui contient l'Object Identifier d'un paquet de « configuration update ». Cette trappe SNMP indique que le capteur attend la requête SNMP envoyée par la Wibox qui va acquitter les paquets de données envoyés par le capteur. Ensuite, le logiciel de la Wibox va envoyer une requête SNMP qui contient différentes informations comme le champ community, qui contient ici le texte GSN\_SET et qui est le principal élément sécuritaire de la version 1 du protocole. Elle contient aussi deux variables, la première contient l'OID de l'acquittement des paquets (1.3.6.1.4.1.28295.99.1.2.1.52.5) et la valeur 7767. Cette valeur est le numéro du paquet acquitté. La deuxième variable correspond à l'OID EAP\_TLS\_SEPARATOR et est nécessaire lors de chaque acquittement de paquet. Une fois que le capteur a reçu la requête, elle la quitte en renvoyant une réponse SNMP contenant les deux mêmes variables avec les mêmes valeurs si tout s'est bien passé. Si une erreur venait à se produire, les champs error-status et error-index contiendraient les différentes informations sur l'erreur survenue.

Afin d'implémenter l'acquittement des paquets dans le programme de communication avec les capteurs, j'ai tout d'abord du capturer et analyser les paquets de données qui arrivent, par défaut sur le port 163. Pour ce faire j'ai créé un thread nommé data\_listener qui ouvre un socket écoutant sur ce port et met les données reçues dans un buffer. La taille de ce buffer est de 1209 bytes ce qui correspond à la taille maximale des paquets de données. En effet, la taille des paquets de données varie, car ils peuvent contenir plusieurs mesures à l'intérieur d'un même paquet.

Afin de récupérer tous les PDUs de données contenus dans chaque paquet, j'ai créé une boucle for qui va parcourir tout le buffer en sélectionnant chaque fois un nouveau PDU. Chaque PDU sélectionné est mis dans un autre buffer temporaire afin de pouvoir en extraire les différentes informations et en particulier le numéro du paquet de données à acquitter. Une fois que toutes les données des mesures ont été extraites du paquet, il va falloir envoyer le numéro du prochain paquet à recevoir ce qui va acquitter la réception des paquets précédent. Pour ce faire, j'ai mis ce numéro dans une structure nommée Heartbeat et qui sera expliquée en détail dans la suite de ce rapport. La valeur de cette variable va être encapsulée dans une requête SNMP qui va être envoyée au capteur par un autre thread nommé snmp\_listener. Chaque donnée de mesure va ensuite devoir être insérée dans la base de données ce qui va également être expliqué dans la suite du rapport.

Par la suite, j'ai dû encore extraire différentes informations du paquet de données qui va être reçu par la Wibox comme le timestamp et le numéro de capteur qui a envoyé le paquet.

## 2.8 Récupération de trappes SNMP :

Il n'est possible d'acquitter les paquets et de reconfigurer les capteurs seulement lorsque ce dernier envoie des trappes SNMP correspondant aux messages de type «Configuration update ». Ceci est fait afin que les capteurs consomment le moins de puissance possible. Par conséquent, afin d'acquitter les paquets de données auprès du capteur, il est nécessaire de vérifier en permanence si une trappe a été reçue et, si c'est le cas, envoyer immédiatement une requête avec les variables correspondantes.

Pour ce faire, j'ai utilisé un logiciel nommé snmptrapd qui est fourni avec l'utilitaire Net-SNMP est qui est, par conséquent, un logiciel open-source. Cet utilitaire est en écoute sur le port SNMP (162) et permet d'effectuer différentes actions lors de la réception d'une trappe.

Les actions à réaliser peuvent être paramétrées suivant l'OID contenu dans la trappe reçue. Pour configurer ces actions il faut éditer le fichier snmptrapd.conf se trouvant dans le répertoire /usr/local/share/snmp sous Linux, après l'installation de la librairie SNMP.

```
disableAuthorization yes
authCommunity log,execute,net public
traphandle .1.3.6.1.4.1.28295.1.* /bin/bash /
/home/trickster888/Desktop/Travail_de_bachelor/scripts/test.sh
```

Figure 25 Exemple de fichier snmptrapd.conf

Dans l'exemple de fichier ci-dessus, on constate que lorsque l'on reçoit une trappe contenant un OID commençant par 1.3.6.1.4.1.28295.1 on va exécuter le script nommé test.sh. L'OID ci-dessus est un OID générique qui correspond à plusieurs équipements d'Aginova. Le script, nommé ici test.sh, va établir une communication socket TCP sur l'adresse localhost (127.0.0.1) et sur le port 1337. Une fois la communication établie, un message qui signifie qu'une trappe a été reçue va être envoyé. Cette méthode permet de répondre rapidement à une trappe SNMP envoyée par le capteur et ainsi de pouvoir reconfigurer le capteur ou acquitter les paquets de données pendant la seconde où le capteur est en écoute sur son port SNMP.

```
#!/bin/bash

# configuration
HOST="127.0.0.1"
PORT="1337"

# define functions
socksend ()
{
    SENDME="$1"
    echo "sending: $SENDME"
    echo -ne "$SENDME" >&5 &
}

sockread ()
{
    LENGTH="$1"
    RETURN=`dd bs=$1 count=1 <&5 2> /dev/null`
}

echo "trying to open socket"
# try to connect
if ! exec 5<> /dev/udp/$HOST/$PORT; then
    echo "`basename $0`: unable to connect to $HOST:$PORT"
    exit 1
fi
echo "socket is open"

# send request
socksend "TEST"

# read 7 bytes for "success"
#sockread 7|
echo "RETURN: $RETURN"
```

Figure 26 Script test.sh

Du côté du programme de communication avec les capteurs, un thread nommé `trap_listener` va être créé afin d'écouter les messages adressés au port 1337 du localhost (le port où le script va envoyer ses informations). Lorsqu'une trappe va être reçue sur ce port, un mutex va alors pouvoir être relâché afin de pouvoir signaler à différents autres threads la réception d'une trappe SNMP. Une fois le mutex libéré, le thread attendant sur ce mutex pourra envoyer les différentes informations SNMP qui étaient en attente au capteur.

A la place d'utiliser ce mécanisme, il aurait été aussi possible de lancer un thread écoutant directement sur le port SNMP (162) et de relâcher directement le mutex de la même façon. J'ai préféré choisir la méthode utilisant le programme `snmptrapd`, car ce programme est très efficace pour analyser les trappes SNMP. Ce qui est très intéressant c'est qu'il est possible de lancer des scripts différents en fonction de l'OID contenu dans la trappe. Ceci donne beaucoup de flexibilité au programme.

Il est aussi possible d'envoyer différents paramètres à un script à la réception d'une trappe, par exemple pour pouvoir envoyer une information différente à l'intérieur du socket en fonction de l'OID présent dans la trappe SNMP. Tout ceci est configurable grâce au fichier `snmptrapd.conf`.

### 2.8.1 Changement dans la réception des trappes SNMP :

J'ai utilisé la version ci-dessus (avec le logiciel `snmptrapd` et le script `test.sh`) pendant une longue partie du développement du logiciel de communication avec les capteurs. Puis après une discussion avec Karl sur les paramètres à stocker dans la base de données, j'ai remarqué que je devais également devoir lire les données présentes dans les paquets de « heartbeat » qui sont eux aussi encapsulés dans des trappes SNMP.

Comme il n'est pas possible d'écouter sur le même port avec deux logiciels différents, j'ai dû arrêter de capturer les trappes SNMP à l'aide de l'utilitaire `snmptrapd`, comme décrit ci-dessus, et modifié le thread `trap_listener` afin que celui-ci écoute sur le port 162 (port de destination des trappes SNMP). Il est maintenant nécessaire de déterminer l'OID contenu dans la trappe SNMP afin de savoir si celle-ci contient un message de type « Configuration update » ou alors si c'est message de type « Heartbeat ». Dans le premier cas, nous relâchons le mutex comme dans la version précédente afin de signaler à différents autres threads la réception d'une trappe SNMP et permettre l'envoi de requête de configuration. Dans le cas où l'on reçoit une trappe de type « Heartbeat », ce dernier va être lu et ses informations extraites comme expliqué dans la section suivante.

## 2.9 Récupérations des paquets « Heartbeat » :

Après avoir récupérer des trappes SNMP de type « Configuration update », il faut aussi pouvoir récupérer les trappes SNMP contenant les informations des paquets de heartbeat. Les informations contenues dans ces paquets vont être stockées dans la base de donnée ainsi qu'envoyées au Portal. Le détail des informations présentes dans ce paquet est détaillé dans le document « Sensor communication-wiki.docx ».

Pour pouvoir récupérer ces informations il faut tout d'abord capturer une trappe SNMP arrivant sur le port 162. Si cette trappe n'est pas une trappe de type configuration update mais bien une trappe de type heartbeat, il va falloir extraire les informations contenues dans ce dernier. Afin de stocker ces informations, j'ai créé une structure nommée `Heartbeat` et contenant les informations utiles présentes dans ces paquets :

```
// Structure contenant les informations de heartbeat
struct Heartbeat{
    char mote_id[5];
    char data[20];
    long int timestamp;
    int data_index;
    int data_id;
    char ack_number[10];
    char voltage[5];
    char clock[10];
    char count[5];
    char rssi[4];
    char uptime[8];
    char resets[4];
    char prodnum[10];
    char WLANWatchdog[6];
    char mac_address[18];
    char strandfwd[2];
    char changeParameterNumber[2];
    char softwareNumber[6];
    char APMacAdress[18];
    char GR1[12];
    char GR2[12];
    char GR3[12];
    char GR4[12];
    char SAF_size[8];
    char AP_reconnect[5];
    char channel[3];
};
```

Figure 27 Structure de type Heartbeat

Ensuite, j'ai créé une fonction que j'ai appelée `getinfos()` à laquelle je vais passer un string contenant les données brutes du paquet ainsi qu'un pointeur sur une structure de type `Heartbeat`. Cette fonction va « parser » les données contenues dans le paquet et les insérer dans les différents champs de la structure. Les différentes données sont chaque fois séparées par les caractères '=' et '&' dans le texte contenu dans le paquet heartbeat. J'ai donc pris le texte présent entre ces deux caractères afin de remplir les différents champs de la structure `Heartbeat`. Ces données sont très importantes et seront stockées dans la base de données de la Wibox comme expliqué dans la suite de ce rapport.





| Size in bytes | Description   |
|---------------|---|
| 4 bytes       | Mote ID   |
| 4 bytes       | New clock value (using mote time)   |
| 4 bytes       | Heartbeat period (in tenth of second)   |
| 4 bytes       | Sampling period (in tenth of second)  |
| 4 bytes       | ACK signal (first data index is 1)  |
| 1 byte        | Control flags, bits: LSB:<br><br>0: control (0=no modifications, 1=modifications),<br><br>1: buzzer (0=off,1=on),<br><br>2: storage strategy(0=overwrite [default], 1=block),<br><br>3-4: connection mode (0=Live mode , 1=Ultra low power)<br><br>5-6: Use for the roll back<br><br>7: Used to empty the internal buffer<br><br>8: Not used yet, reserved for future usage |
| 2 bytes       | Product number (small format)   |
| 2 bytes       | Channels scan bits (low bit is channel 1, high bit is channel 16 ie $1 < 16$ )  |
| 2 bytes       | Power hold time (in ms), only valid for the WLS   |
| 2 bytes       | New Sample sending rate   |
| 1 bytes       | Test the module (1=test needed) only available for the SET  |
| 4 bytes       | GR1 timestamp only available for the SET  |



|             |  |
|-------------|--|
| 4 bytes     | GR2 timestamp only available for the SET |
| 4 bytes     | GR3 timestamp only available for the SET |
| 4 bytes     | GR4 timestamp only available for the SET |
| other bytes | Ignored (forward compatible)             |

On constate qu'il est possible de changer plusieurs paramètres grâce aux différents champs contenus dans la variable liée à l'OID « vendor specific ». Pour changer la sampling period, il est nécessaire de modifier les bytes 13, 14, 15 et 16 de la variable en y ajoutant la valeur désirée en dixième de secondes.

Afin de pouvoir modifier les différents paramètres liés au vendor specific, j'ai créé un programme nommée server\_tcp. Ce programme va utiliser le protocole TCP afin de pouvoir transmettre des données au programme de la Wibox lite communiquant avec les capteurs. Pour ce faire, j'ai créé une structure comprenant un numéro de commande, un OID (en l'occurrence celui du vendor specific) et une variable contenant les différents paramètres fixé par l'utilisateur du programme.

```
struct Commande{
    unsigned int id_capteur;
    unsigned int end;
    unsigned int numero;
    unsigned int reset;
    char oid[MAX_OID_LEN];
    char variable[TAILLE_VARIABLE];
    char type;
    struct Commande * pSuivant;
};
```

Figure 29 Structure d'une commande

Un numéro contenant l'ID du capteur est présent dans la base de données, ce numéro n'est pour l'instant pas encore utilisé dans cette version de la Wibox lite mais le sera peut-être dans une version future devant gérer plusieurs capteurs à la fois.

Le numéro de la commande sert à définir si la commande est une commande standard ou non. Si la commande n'est pas standard, cela signifie qu'il faut envoyer une commande avec des valeurs par défaut avant d'envoyer les nouvelles commandes de reconfiguration. Ceci est géré dans le programme de la Wibox lite. Le champ OID définit, comme son nom l'indique l'OID de la commande et le champ variable contient une variable liée à cet OID. Le champ reset est par défaut à 0 mais s'il est à 1, cela signifie qu'une commande contenant l'OID du reset du capteur doit être envoyée. Ceci est utilisé notamment lors du changement de l'adresse IP du capteur. Le champ type de la commande sert à définir le type de la variable. Celle-ci peut être du type « OctetString »,

« Integer32 » ou « IPAddress ». Le dernier champ nommé pSuivant est un pointeur sur la prochaine commande. Il sert à chaîner les commandes afin de pouvoir en envoyer plusieurs à la fois.

Le programme doit tout d'abord initier une communication socket utilisant le protocole TCP sur le port 4098 du localhost. Une fois que la connexion a été établie, un menu va être affiché à l'utilisateur qui va pouvoir choisir quelle commande exécuter. Si l'utilisateur choisit, par exemple, d'exécuter une commande changeant la sampling period, c'est la valeur de l'OID vendor specific qui va être affecté au champ oid de la structure. Ensuite, toujours pour un changement de sampling period, la valeur entrée par l'utilisateur va être multipliée par 10 puis affectée aux 4 bytes réservés pour la valeur de la sampling period.

```
// Set sampling period
case 1:
    strcpy(commande->oid, vendor_specific);

    // Specifie le type de la commande
    commande->numero = 0;

    // Specifie le type de la commande
    commande->type = 'x';

    // Conversion de la valeur en dixieme de secondes
    valeur_commande *= 10;

    // Conversion de la valeur en hexadecimal
    sprintf(string_command, "%.2x", valeur_commande);

    // Ajout de la valeur dans la variable a envoyer
    vendor_specific_variable[SAMPLING_PERIOD] = string_command[0];
    vendor_specific_variable[SAMPLING_PERIOD + 1] = string_command[1];

    // Affectation de la valeur à la commande
    strcpy(commande->variable, vendor_specific_variable);
    break;
```

Figure 30 Affectation des différents champs d'une commande

On constate que la valeur associée au type de la commande contient la valeur « x » c'est-à-dire que la variable sera du type « OctetString ». Le champ pSuivant est pour chaque commande initialisé à NULL. Il n'est donc pas nécessaire de s'occuper de ce champ lors de l'envoi d'une seule commande. Afin de convertir la valeur en hexadécimal, j'ai utilisé la fonction « sprintf » qui va prendre la valeur entrée par l'utilisateur, la convertir en hexadécimal et l'affecter à la variable string\_command. Il faut maintenant affecter cette valeur à l'emplacement prévu de la sampling\_period et pour finir affecter la variable du vendor specific à la variable de la commande.

La modification de la sampling period consiste en l'envoi d'une commande. Par contre, pour changer l'adresse IP de la Wibox par exemple, il est nécessaire d'envoyer trois commandes. Pour ce faire, il est nécessaire de créer deux commandes dynamiquement à l'aide de la fonction malloc(). Cette

fonction réserve dynamiquement un espace mémoire d'une taille passée en paramètre et retourne un pointeur sur cet espace mémoire. Une fois que ce pointeur est récupéré, il faut chaîner les commandes, c'est-à-dire qu'il faut affecter le champ pSuivant de la première commande et ainsi de suite. La valeur du champ pSuivant de la dernière commande doit avoir la valeur NULL afin de signaler que c'est la dernière de la liste.

```
// Deuxieme commande a envoyer
struct Commande * deuxieme_commande = malloc(sizeof * deuxieme_commande);
// Troisieme commande a envoyer
struct Commande * troisieme_commande = malloc(sizeof * troisieme_commande);

// chainage des commandes
commande->pSuivant = deuxieme_commande;
deuxieme_commande->pSuivant = troisieme_commande;
troisieme_commande->pSuivant = NULL;
```

Figure 31 Création et chainage des différentes commandes à envoyer

Une fois que la structure a été complétée, la liste contenant les différentes commandes va être envoyée. Pour envoyer complètement la liste, le programme va commencer par envoyer la première commande, puis tant que le champ pSuivant de la commande ne vaut pas NULL, il va envoyer les commandes suivantes. Une fois les commandes envoyées, il est nécessaire de libérer l'espace mémoire réservé par les commandes. Cette opération se fait grâce à la fonction free() à laquelle on passe en paramètre un pointeur sur la zone mémoire à libérer.

Du côté du programme de communication avec les capteurs, un thread nommé command\_listener est chargé d'écouter le port sur lequel les commandes sont envoyées. Lorsqu'une commande est reçue, elle va être mise dans une queue de commandes. Il est nécessaire de reconstituer la liste de commande en réservant dynamiquement de la mémoire car, comme les deux programmes ne font pas partie du même processus, ils ne peuvent partager un même espace mémoire. Toutes les commandes se trouvant dans cette queue vont être envoyées au capteur lors de la réception de la prochaine trappe SNMP.

Pour recevoir ces différentes commandes et constituer une liste chaînée, j'ai créé un pointeur sur une structure commande que j'ai appelé tete\_liste. Le pointeur tete\_liste va pointer sur la première commande de la liste chaînée. Pour distinguer la première commande des suivantes, j'ai utilisé une variable de type integer nommée premiere\_commande et qui est initialisée à 0. Lorsque la première commande a été reçue, cette variable passe à 1 et elle est remise à zéro lorsque la chaîne de commande a été envoyée. Ensuite toutes les autres commandes reçues vont être chaînées à la suite de la première commande. Pour pouvoir chaîner les différentes commandes, j'ai utilisé une fonction nommée void ajouterElement(struct Commande \*\* tete\_liste, struct Commande \* commande) qui prend en paramètre le pointeur sur la première commande de la liste ainsi que la commande à ajouter en queue de liste. Pour ce faire il est nécessaire de réserver dynamiquement de l'espace mémoire grâce à la fonction malloc() et ensuite, il faut parcourir toute la liste afin de chaîner la commande en bout de liste. Voici le code de cette fonction :

```

void ajouterElement(struct Commande ** tete_liste, struct Commande * commande)
{
    // On crée un nouvel élément
    struct Commande * nouvelle_commande = malloc(sizeof(struct Commande));

    // On assigne la valeur au nouvel élément
    *nouvelle_commande = *commande;

    // On ajoute en fin, donc aucun élément ne va suivre
    nouvelle_commande->pSuivant = NULL;

    // Sinon, on parcourt la liste à l'aide d'un pointeur temporaire et on
    // indique que le dernier élément de la liste est relié au nouvel élément
    struct Commande * temp = (*tete_liste);
    while(temp->pSuivant != NULL)
    {
        temp = temp->pSuivant;
    }
    temp->pSuivant = nouvelle_commande;
    nouvelle_commande->pSuivant = NULL;
    return;
}

```

Figure 32 Fonction ajouterElement()

Ensuite, la mémoire allouée dynamiquement lors de la réception des commandes, sera libérée par le thread snmp\_listener une fois que les commandes ont toutes été envoyées. Pour ce faire, il faut parcourir la liste et supprimer toutes les commandes en sauvegardant le pointeur suivant jusqu'à ce que celui-ci contienne la valeur NULL comme dans la figure ci-dessous :

```

if(save->pSuivant != NULL){
    tete_liste = save->pSuivant;
    while(tete_liste != NULL){
        save = tete_liste->pSuivant;
        free(tete_liste);
        tete_liste = save;
    }
}

```

Figure 33 Libération de la mémoire de la liste chaînée

Pour envoyer des commandes lors de la réception d'une trappe SNMP, j'ai dû faire de la programmation concurrente. Pour que le thread qui envoie les différents paquets SNMP attende un message « Configuration update » soit arrivé du capteur j'ai utilisé un mutex que je vais bloquer avant le lancement des différents threads. Ce mutex sera débloqué lors de la réception d'une trappe SNMP contenant l'OID du message de « Configuration update » et il sera à nouveau bloqué juste après l'envoi des commandes SNMP par le thread snmp\_listener.

La partie de reconfiguration des commandes a été longue car, comme il n'existe aucun document expliquant quelles commandes envoyer pour reconfigurer les capteurs, il a fallu analyser chaque

trame pour savoir quelles commandes envoyer au capteur. Le grand avantage du système de commande que j'ai implémenté est que l'on peut très facilement ajouter de nouvelles fonctionnalités au programme de reconfiguration des capteurs ce qui est très utile pour la flexibilité du programme. Je n'ai pas présenté le détail de l'implémentation de chaque commande de reconfiguration dans ce rapport car cela aurait été très long et répétitif de plus, le code est disponible dans le fichier `server_tcp.c`.

### 2.11 Contrôle de saisie :

Comme l'utilisateur peut entrer des commandes qui vont ensuite être envoyées aux capteurs par l'intermédiaire du programme de la Wibox lite, il est nécessaire de contrôler les saisies faites par l'utilisateur. Pour contrôler ces saisies, j'ai créé deux fonctions : `numberEntry()` et `addressEntry()`. Comme leurs noms l'indiquent, `numberEntry()` sert à contrôler la saisie d'un nombre et `addressEntry()` contrôle la saisie d'une adresse IP.

Ces deux fonctions prennent en paramètre un texte ainsi qu'un pointeur sur la zone mémoire à remplir avec la donnée saisie. Dans les deux cas, c'est d'abord le texte qui indique la valeur à entrer à l'utilisateur qui est affiché. Ensuite, pour la saisie d'un nombre, la saisie limite les caractères entrés seulement aux chiffres de 0 à 9. Puis, une fois que le nombre a été saisi et affecté à une variable, je vide le buffer et je contrôle que la valeur saisie soit correcte. Si cette valeur n'est pas correcte, un message d'erreur est affiché et l'utilisateur est invité à entrer une nouvelle valeur.

```
void numberEntry(char * texte, int * valeur){  
  
    int ret = 0;  
  
    while(ret != 1){  
  
        // Affichage du texte  
        printf("%s", texte);  
  
        // Saisie de la valeur  
        ret = scanf("%d[0-9]", valeur);  
        scanf ("%*[^\n]");  
        getchar();  
  
        // Controle la validite du nombre saisi  
        if(ret != 1 || *valeur < 0){  
            ret = 0;  
            printf("Erreur de saisie, entrez une valeur correcte\n");  
        }  
    }  
}
```

Figure 34 Contrôle saisie d'un nombre

Afin de contrôler la saisie d'une adresse IP, j'ai créé un tableau de 4 int. Chaque int va contenir une partie de l'adresse IP que je vais saisir grâce à l'outil de formatage `"%d.%d.%d.%d"` de la fonction `scanf()`. Une fois que l'adresse a été saisie, la fonction contrôle que chaque partie soit positive et

inférieure à 255 qui est la valeur maximale pour un champ d'une adresse IP. Si un de ces test échoue, alors l'utilisateur est invité à entrer une nouvelle adresse IP. Pour finir, la valeur entrée par l'utilisateur est affectée au pointeur passé en paramètre grâce à la fonction `sprintf()`.

```
void addressEntry(char * texte, char * address){

    int ipaddress[4], ret = 1;

    // Affichage du texte
    printf("%s", texte);

    while(ret != 0){
        // Saisie et controle de l'adresse entree
        if(ret = scanf("%d.%d.%d.%d", &ipaddress[0], &ipaddress[1], &ipaddress[2], &ipaddress[3]) != 4 ||
            ipaddress[0]>255 || ipaddress[0] < 0 || ipaddress[1]>255 || ipaddress[1] < 0 ||
            ipaddress[2]>255 || ipaddress[2] < 0 || ipaddress[3]>255 || ipaddress[3] < 0){
            printf("Erreur de saisie, entrez une adresse correcte: ");
        }

        // Vide la buffer
        scanf ("%*[^\\n]");
        getchar();
    }

    sprintf(address, "%d.%d.%d.%d", ipaddress[0], ipaddress[1], ipaddress[2], ipaddress[3]);
}
```

Figure 35 Contrôle de la saisie d'une adresse IP

Maintenant que la partie de reconfiguration des capteurs est terminée, il faut implémenter une base de données qui va contenir différentes mesures et informations envoyées par les capteurs.

## 2.12 Présentation de SQLite

Afin de pouvoir stocker les données envoyées par les capteurs, j'ai utilisé un utilitaire permettant de gérer une base des données embarquée. Cet utilitaire s'appelle SQLite. SQLite est une bibliothèque écrite en C qui propose un moteur de base de données relationnelle accessible par le langage SQL, ce qui est un grand atout pour une base de données embarquée. La licence de ce logiciel est une licence public je peux donc l'utiliser sans problème dans le cadre de mon travail de Bachelor.

SQLite convient très bien à l'utilisation de systèmes embarqués du fait de sa très petite taille (moins de 300 ko). Il est très utilisé sur la plupart des smartphones modernes comme l'iPhone ainsi que dans de nombreux systèmes d'exploitation mobiles comme Symbian et Android. La grande différence entre SQLite et la majorité des bases de données est que SQLite est directement intégré dans l'application qui utilise sa bibliothèque logicielle avec son moteur de base de données. L'accès à une base de données SQLite se fait en ouvrant un fichier qui lui est propre contenant ses déclarations, ses tables et ses index ainsi que ses données.

Le fait que SQLite soit intégré directement dans le programme utilisant la base de données est très intéressant car cela rend l'accès aux données plus rapide, plus sécurisé, plus structuré, plus facile et surtout totalement indépendant de la plateforme utilisée tout en ne portant pas atteinte à la facilité de déploiement de l'application qui l'utilise.



### 2.12.1 Création d'une base de données test :

Afin de bien comprendre les possibilités offertes par l'utilitaire SQLite et pour me familiariser avec son fonctionnement, j'ai décidé de créer une base de données « test » dont voici une description des différentes étapes.

Premièrement, il faut créer la base de données grâce à la fonction `sqlite3_open()` qui prend en paramètre un string contenant le nom de la base de données ainsi qu'un pointeur de type `sqlite3` afin de récupérer l'adresse de la base de données. En cas de problème, une valeur différente de 0 est retournée par cette fonction. Si tout s'est bien passé, je peux maintenant créer les différents champs présents dans la base de données.

Pour créer ces différents champs, il faut créer un string contenant les informations sur la table que l'on désire créer. Afin de stocker les données des capteurs, j'ai d'abord créé une table nommée `Sensor` qui va comprendre un champ nommé « id » qui va être l'identifiant du paquet. Ce champ comportera simplement un chiffre il sera donc du type `integer`. Puis, le second champ va contenir le temps système auquel le paquet a été inséré dans la base de données. Ce champ s'appelle « time » et est de type `long`. Pour finir, le troisième champ se nomme « data » et est de type `string`. Il va contenir les informations brutes envoyées par le capteur. Voici la création de la base de données ainsi que de sa première table :

```
// Recuperation du pointeur sur la base de donnees
rc = sqlite3_open ("sensor.db", &db);
if (rc){
    fprintf (stderr, "Can't open database: %s\n", sqlite3_errmsg (db));
    sqlite3_close (db);
    exit (1);
}

// Creation de la table de la base de donnees
char *requete0 = "CREATE TABLE Sensor (id INTEGER, time LONG, data varchar(32));";
if (sqlite3_exec (db, requete0, callback, 0, &zErrMsg) != SQLITE_OK){
    printf ("Erreur requete0 :(\n");
}
```

Figure 36 Création de la base de données « test »

Une fois que la base de données et la première table de celle-ci créées, le thread `data_listener` va être créé par le thread `main` afin de remplir cette base de données avec les paquets provenant du capteur connecté. A l'intérieur de ce thread, nous parcourons les données contenant les mesures faites par le capteur. Les données sont ainsi stockées à l'intérieur d'un buffer temporaire. Comme je vais devoir stocker trois champs à l'intérieur de la table, je crée une requête contenant trois champs dont les valeurs ne sont pas encore définies. Ceci est possible grâce au caractère « ? » qui permet de fixer la valeur d'un champ ultérieurement. Puis, pour chaque mesure effectuée par le capteur et envoyée à la Wibox, je vais enregistrer dans la base de données l'index de la mesure, le temps auquel elle a été reçue, qui va être récupéré grâce à la fonction `gettimeofday()`, ainsi que les données contenues dans le buffer.

L'insertion de ces différentes données est faite à l'intérieur d'une boucle for présente dans le thread data\_listener qui parcourt les données reçues afin d'en extraire les mesures. Voici le code utilisé afin de réaliser l'insertion dans la base de données :

```
// Parcoure les donnees recues et enregistre sauve les differents paquets
for(i = 0; i < n/TAIILE_PDU_DATA; i++){
    memcpy(buffer_temp, buffer+i*TAIILE_PDU_DATA, sizeof buffer_temp);

    // Sauvegarde du numero du paquet
    temp_prec = temp;

    // Actualisation numero du paquet
    temp = buffer_temp[0] * 256 + buffer_temp[1] + 1;

    printf("temp: %d, temp_prec:%d\n", temp, temp_prec);

    if(temp_prec < temp){
        // Insertion de l'index dans la base de donnees
        sqlite3_bind_int (stmt, 1, index++);

        // Recuperation du temps courant et insertion dans la base de donnees
        gettimeofday(&now, NULL);
        sqlite3_bind_int (stmt, 2, now.tv_sec);
        sqlite3_bind_text (stmt, 3, buffer_temp, 34, NULL);

        // Insertion des donnees dans le base de donnees
        step(sqlite3_step(stmt));
        sqlite3_reset (stmt);
    }
}
```

Figure 37 Insertion des données dans la table

On constate dans le code ci-dessus que la fonction `sqlite3_bind_int()` prend la requête SQL qui va être insérée dans la base de donnée et complète le premier paramètre par l'index. Puis, le deuxième paramètre, qui est le temps courant, va être ajouté à la table grâce à la même requête. Ensuite c'est la fonction `sqlite3_bind_text()` qui va être appelée pour insérer les données contenues dans le buffer temporaire dans la base de données. Une fois que la requête est complétée avec les bons paramètres, elle est insérée dans la table grâce à la fonction `step(sqlite3_step(stmt))` puis les données contenues dans la requête sont remises à zéro par la fonction `sqlite3_reset(stmt)`.

## 2.13 Création de la base de données réelle

La base de données que je vais créer pour ce projet comporte deux tables et se nommera Wibox.db. Une table comportant les informations relatives au capteur comme son id, son adresse MAC et son product number et la deuxième table comportant les données envoyées par le capteur. J'ai nommé la première table comportant les informations sur le capteur « Sensor\_information » et la deuxième table contenant les données du capteur « Sensor\_data ». La table Sensor information comportera une entrée pour chaque capteur tandis que la table contenant les données des capteurs comportera



un nombre bien plus élevé d'entrées. Il sera aussi nécessaire de stocké, dans la table Sensor\_data, le temps auquel les données ont été reçues comme cela a déjà été fait dans l'exemple précédent.



Figure 38 Schéma de la base de données relationnelle

Comme nous pouvons le constater sur le schéma de la base de données, la clé primaire de la table Sensor\_information est le champ mote\_id. Ce champ contient le numéro du capteur qui est unique, c'est pourquoi il peut servir de clé primaire. La clé primaire de la table Sensor\_data est également le champ mote\_id, grâce à cela, on peut effectuer des requêtes SQL qui mettent en relation les informations du capteur ainsi que ses données. La base de données est donc bien relationnelle.

Pour créer cette base de données, j'ai tout d'abord créé les tables à l'aide des commandes suivante :

```

// Creation de la table de la table Sensor_informations
char *requete0 = "CREATE TABLE Sensor_informations (mote_id INTEGER, voltage varchar(5),\
softwareNumber varchar(6), mac_address varchar(18), PRIMARY KEY (mote_id));";
if (sqlite3_exec (db, requete0, callback, 0, &zErrMsg) != SQLITE_OK){
    printf ("Erreur requete0 :(\n");
}

// Creation de la table Sensor_data
char *requete1 = "CREATE TABLE Sensor_data (mote_id INTEGER, time LONG, timestamp LONG,\
data_id INTEGER, data varchar(32), FOREIGN KEY(mote_id) REFERENCES Sensor_informations(mote_id));";
if (sqlite3_exec (db, requete1, callback, 0, &zErrMsg) != SQLITE_OK){
    printf ("Erreur requete1 :(\n");
}
  
```

Figure 39 Création des tables de la base de données

Les tables sont créées dans le thread principal (main) et ceci est fait avant le lancement des threads car ce sont les différents threads qui vont remplir la base de données. Plus précisément, c'est le thread nommé data\_listener qui va remplir la table Sensor\_data et c'est le thread trap\_listener qui va remplir la table Sensor\_informations.

### 2.13.1 Sensor\_data :

Les données sont récupérées dans le thread data\_listener qui écoute les données arrivant sur le port 163. Ensuite, ces données vont être parcourues par le programme qui va premièrement en extraire le mote\_id, cette valeur correspond au capteur qui a envoyé les données et va être insérée dans la base de données car c'est la clef étrangère de cette table. Comme le mote\_id ou sensor\_id est également stocké dans la table Sensor\_informations comme clé primaire, les deux tables sont liées par cet identifiant.

Maintenant que l'identifiant a été inséré dans la base de données, c'est le numéro de paquet qui va être extrait du paquet de données. Ce numéro de paquet est une donnée importante car c'est ce numéro qui va être envoyé au capteur afin d'acquitter le paquet de donnée reçu. Une fois que ce numéro a été extrait il est nécessaire d'effectuer une conversion de celui-ci afin de l'enregistrer en base décimale dans la base de données.

Ensuite, c'est le temps courant récupéré avec la fonction `gettimeofday()` qui va être inséré dans la base de données. Il est important de savoir quand les données ont été reçues afin que les données les plus anciennes puissent être effacées périodiquement. L'information timestamp qui est aussi envoyée par les capteurs est enregistrée dans la base de données, cette information permet de savoir quand les données ont été envoyées par le capteur. Grâce à ces deux informations de temps, on est maintenant capable de savoir quand les données ont été envoyées du capteur et reçues par la Wibox Lite.

Pour finir, les données envoyées par le capteur vont être parsées et converties sous forme de texte. En effet, dans le paquet UDP reçu par la Wibox lite, les données sont uniquement en hexadécimal, il est donc nécessaire de les convertir sous forme de texte afin de pouvoir les lire beaucoup plus facilement dans la base de données. Pour convertir les données en hexadécimal, j'ai utilisé une boucle while qui va parcourir les données brutes et les convertir en hexadécimal à l'aide de la fonction `sprintf` qui permet de formater une chaîne de caractère, voici l'algorithme de formatage ci-dessous :

```
// Conversion des donnees dans le format char
while(index != OFFSET_PAYLOAD_DATA + TAILLE_PAYLOAD_DATA) {
    sprintf(sensor_data, "%02x", (char)buffer_temp[index++]);
    sensor_data+=2;
}
sensor_data = p;
index = OFFSET_PAYLOAD_DATA;

// Insertion des donnees du capteur dans la base de donnees
sqlite3_bind_text (stmt, 4, sensor_data, 17, NULL);
```

Figure 40 Conversion des données au format char

### 2.13.2 Sensor\_informations :

La table `Sensor_informations` est remplie par les informations qui sont extraites des trappes de heartbeat qui sont envoyées par le capteur. J'ai créé une fonction nommée `getInfos()` qui va décoder cette trappe et en extraire les informations afin de les mettre dans une structure propre au heartbeat. Maintenant que les informations sont récupérées, il ne reste plus qu'à les insérer dans la base de données comme nous le montre le code ci-dessous :

```
// Insertion de l'index dans la base de donnees
sqlite3_bind_int (stmt, 1, atoi(heartbeat.mote_id));
// Insertion du voltage dans la base de donnees
sqlite3_bind_text (stmt, 2, heartbeat.voltage, 5, NULL);
// Insertion du numero de produit dans la base de donnees
sqlite3_bind_text (stmt, 3, heartbeat.prodnum, 10, NULL);
// Insertion de la mac address dans la base de donnees
sqlite3_bind_text (stmt, 4, heartbeat.mac_address, 17, NULL);
step(sqlite3_step(stmt));
sqlite3_reset (stmt);
```

Figure 41 Insertion des informations dans la database

## 2.14 Nettoyage de la base de données :

Comme nous ne voulons pas que la base de données ne prenne pas trop de place en stockant des données obsolètes, j'ai créé une fonction qui va être appelée tous les X temps (temps paramétrable par l'utilisateur) et qui va effacer les données obsolètes présentes dans la base de données. Pour ce faire, j'ai créé une fonction qui va lire les différentes informations entrées par l'utilisateur dans le fichier config.txt qui doit être placé dans le répertoire /etc/ sous Linux (ce répertoire contient les fichiers de configuration de tous les programmes installés sous Linux).

Dans ce fichier, deux options sont disponibles : le réglage de la fréquence à laquelle le programme va contrôler l'état des données à l'intérieur de la base de données et le temps depuis lequel les données sont jugées comme étant obsolètes. Le programme va donc parcourir toute la base de données lorsque le temps de contrôle est atteint et va effacer toutes les données dont le temps de rétention est supérieur aux temps considéré obsolète par l'utilisateur de la Wibox lite.

Si aucun fichier de configuration n'est disponible, un temps par défaut va être assigné au contrôle de la base de données. Par défaut, le programme va aller lire toute la base de données une fois par jour et les données seront considérées comme étant obsolètes si elles ont plus d'une semaine.

## 2.15 Lecture de la base de données :

Pour pouvoir lire la base de données SQLite que je viens de créer, j'avais tout d'abord écrit un petit programme qui lisait les données brutes de la base de données et les affichait à l'écran. Puis, j'ai pensé qu'il serait beaucoup plus simple de télécharger un utilitaire qui puisse lire les bases de données SQLite et les afficher en détail afin que l'utilisateur de la Wibox lite puisse les parcourir et vérifier rapidement les données capturées.

J'ai donc téléchargé un logiciel nommé « SQLite Database browser » et disponible à l'adresse suivante : <http://sourceforge.net/projects/sqlitebrowser/>. Ce logiciel, conçu pour Microsoft Windows, est très simple d'utilisation et ne nécessite aucune installation, il suffit juste de lancer l'exécutable afin de démarrer le programme.

Une fois le programme ouvert, plusieurs fonctionnalités sont disponibles, il est possible, par exemple, de créer une base de données SQLite mais aussi d'ouvrir une base de données déjà existante afin de voir la structure de celle-ci ou encore d'exécuter des requêtes SQL sur cette dernière. Voici les

différentes options possibles lors de l'ouverture d'une base de données avec SQLite Database browser :

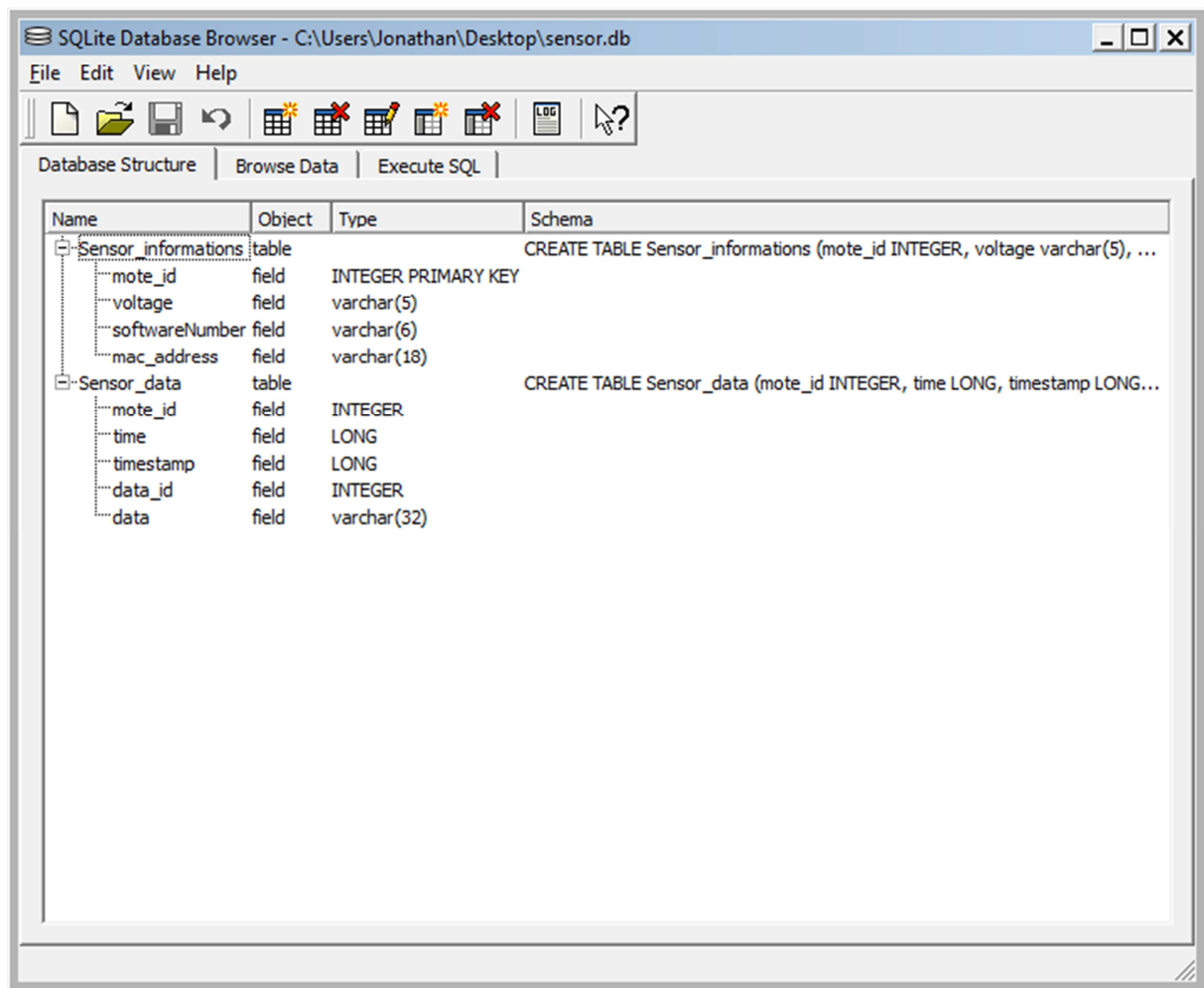


Figure 42 Structure de la base de données dans SQLite Database browser

Nous pouvons, dans cet onglet, analyser la structure de la base de données, c'est-à-dire le type des différents champs et la composition de ceux-ci. Cette option est très pratique et permet de savoir rapidement si la structure de la base de données a été correctement implémentée ou non. Ensuite nous avons l'onglet suivant qui s'intitule « Browse Data » :

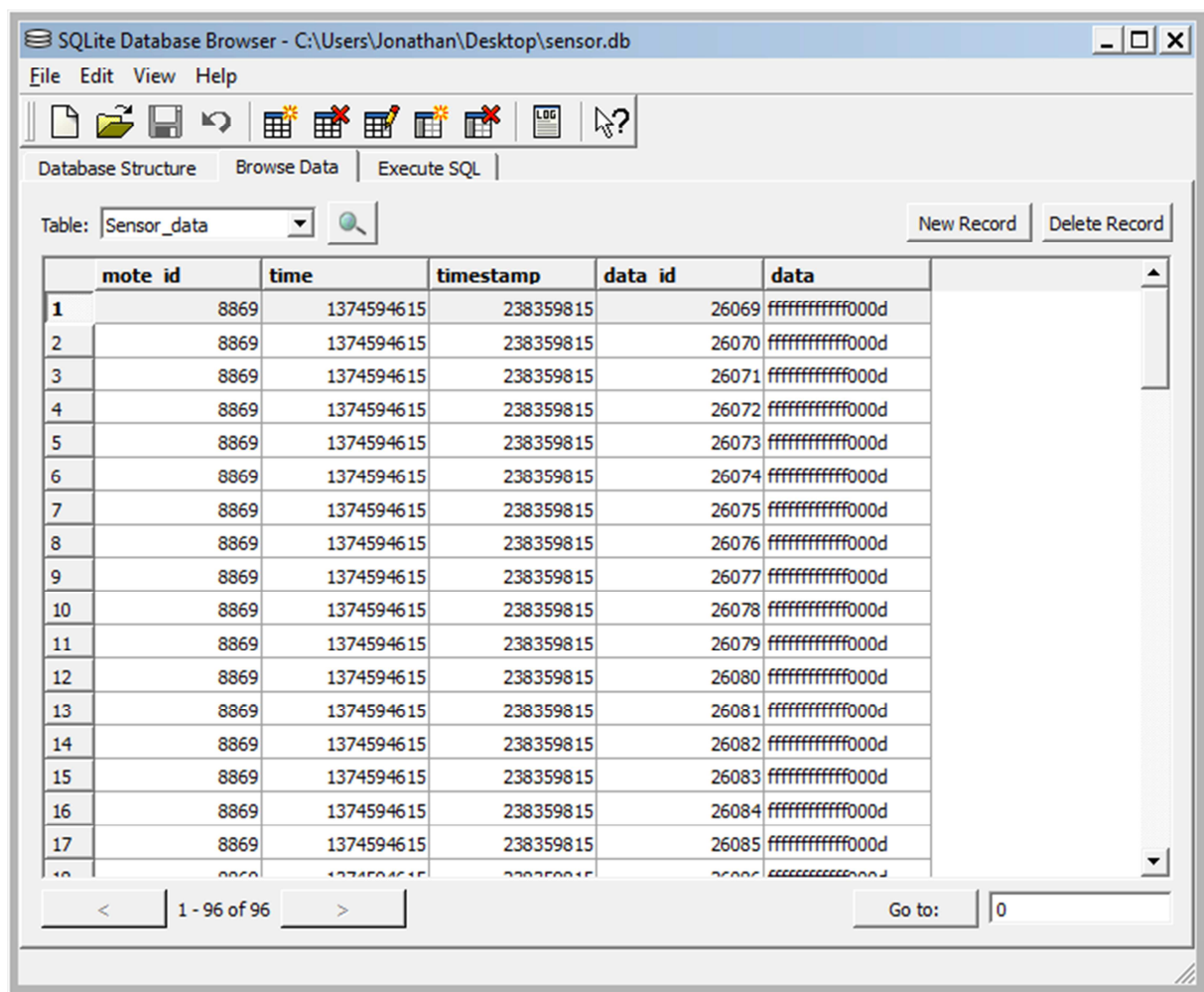


Figure 43 Onglet Browse Data

Dans cet onglet, nous pouvons apercevoir le contenu des différents champs qui ont été entrés dans la base de données. Nous constatons que les premiers paquets ont tous le même temps de réception, ceci vient du fait que lorsque le capteur se connecte à la Wibox, il envoie tous les paquets qui n'ont pas été acquittés en même temps auprès de la Wibox qui les enregistre donc avec le même temps dans la base de données. Les données envoyées par le capteur sont au format hexadécimal j'ai donc dû les convertir en caractère afin de pouvoir les insérer dans la base de données.

Le dernier onglet est l'onglet Execute SQL :

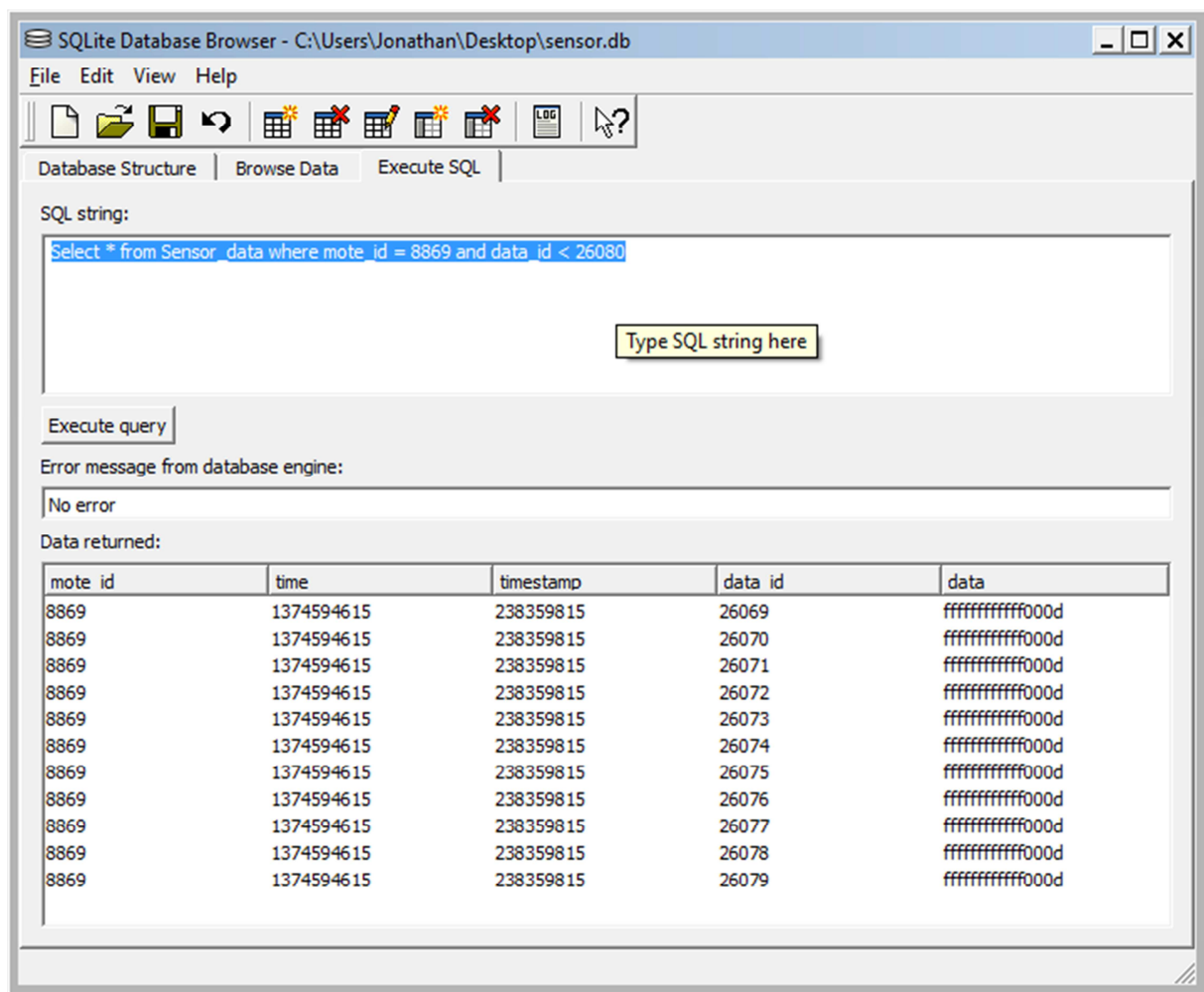


Figure 44 Onglet Execute SQL

Comme son nom l'indique, c'est à l'aide de cet onglet que nous pouvons effectuer des requêtes SQL afin d'afficher les données que nous sélectionnons dans la base de données. Ceci est une fonctionnalité très utile que fournit ce programme car les requêtes SQL sont extrêmement utilisées dans le monde des bases de données. Pour faire un exemple de requête SQL, j'ai choisi d'afficher tous les paquets envoyés par le capteur 8869 dont le data\_id est inférieur à 26080 d'où la requête : `Select * from Sensor_data where mote_id = 8869 and data_id < 26080`.

### 3 Communication avec le Portal

#### 3.1 Types de paquets envoyés au Portal :

Une fois que la communication sans fil entre la Wibox et les capteurs a été implémentée, la deuxième grosse partie de ce travail est d'effectuer la communication entre la Wibox Lite et le Portal. Pour rappel, le Portal sert à afficher les informations sur les capteurs à l'utilisateur final.

La communication entre le Portal et la Wibox se fait des deux sens et en utilisant deux protocoles différents qui sont le XML-RPC et le XML/http.

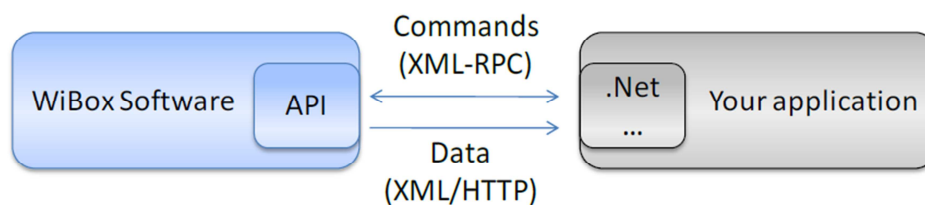


Figure 45 Protocole de communication

Le protocole XML-RPC est utilisé pour envoyer des commandes et recevoir des réponses immédiates de la Wibox (par exemple lors de la reconfiguration d'un capteur). Tandis que le protocole XML/http est utilisé par la Wibox pour envoyer un flux de données ainsi que d'autres informations asynchrones au client.

Afin de recevoir des données, le client doit établir une session permanente et authentifiée avec la Wibox. La connexion (avec authentification) est établie en utilisant l'URL suivant : `http://hostname :port/data`.

Où :

- Hostname : Adresse IP de la machine sous laquelle fonctionne la Wibox.
- Port : 8088\*
- Username : ws
- Password : ws79013\*

(\*) Valeurs par défaut, configurables.

Une fois que la connexion est établie, la Wibox va envoyer un flux de données continu dans le format ci-dessous :

```
<?xml version="1.0" encoding = "UTF-8"?>
<Messages>
...
...
```

Figure 46 Exemple de message XML/http



Le flux de données se poursuivra tant que la Wibox n'est pas redémarrée, éteinte ou que la communication n'est pas interrompue. Il est important de noter que l'élément « Messages » ne sera jamais fermé. Les éléments XML suivants se trouveront en dessous de l'élément racine « Messages ».

- Health : La Wibox envoie des messages continus afin de montrer qu'elle fonctionne toujours (et que la connexion entre la Wibox et le Portal fonctionne également).
- HaveData (aka heartbeats) : Ces paquets sont envoyés par chaque capteur à un intervalle de temps configurable. Les Heartbeats contiennent des informations sur les capteurs comme l'ID, l'adresse MAC, le voltage, etc...
- Linkup : Envoyés aussi par chaque capteurs à un intervalle de temps configurable, ces paquets contiennent simplement l'ID du capteur afin d'indiquer que le capteur fonctionne toujours.
- Data : Données envoyées par les capteurs. Les données sont stockées dans la mémoire des capteurs afin de conserver toutes les données en sécurité.
- WiboxReset : Envoyés lorsque au démarrage de la Wibox. Ce type de paquets est utilisé afin de rafraichir l'état du client.
- Update : Lorsqu'une commande est envoyée à un capteur à travers le Portal, un nombre unique est utilisé afin de suivre les messages individuels. Une mise à jour est envoyée par la Wibox pour les longues taches (par exemple la reprogrammation) afin de garder le client informé.
- Notification : Même chose que les updates, mais envoyés à la fin de la reconfiguration avec un champ de statut succès/échoué.
- Error : Paquets envoyés en cas d'erreur.

Voici un exemple typique de données XML reçues de la Wibox :

```
<?xml version="1.0" encoding = "UTF-8"?>
<Messages>
  <HaveData product="XTEMP-1006-0002" isReader="false" mote id="5215"
heartbeat="10" sampling="10" time="89310538" codeVersion="G5.2.1"
mac="00:1d:c9:0a:00:d7" voltage="34" rssi="-41" config="20"
storeFwdEnabled="1" upTime="176768" count="0" parent="0"></HaveData>
  <Data mote id="5215" data index="20534" data id="28776"
timestamp="89306324" data_type="1" data="032cffffffff0003" />
  <Health voltage="0" sourceaddr="0" RSSI="0" originaddr="0" seqno="0"
quality="0"></Health>
```

Figure 47 Paquets envoyé de la Wibox au Portal.

Dans l'exemple ci-dessus, la Wibox, après avoir ouvert le tag « Messages », envoie un message de type « HaveData » qui signale qu'un capteur a envoyé un paquet « Heartbeat ». Le second message présent envoyé est un message contenant des données envoyées par le capteur. Le troisième message présent dans cet exemple est un message de Linkup, utilisé pour maintenir la connexion entre la Wibox et le Portal.



### 3.2 Implémentation de la communication avec le Portal :

Sur ma machine, la version complète de la Wibox ainsi que le Portal fonctionnent tous deux sous Windows. Comme ces deux logiciels fonctionnent sur la même machine et sur le même OS, ils communiquent en envoyant leurs données sur l'adresse localhost (127.0.0.1). Sous Windows 7, il n'est malheureusement pas possible de capturer et d'analyser les paquets transitant par l'adresse localhost à l'aide de Wireshark. J'ai donc dû utiliser un logiciel nommé RawCap qui permet de capturer les paquets transitant par le localhost et dont le fonctionnement est décrit dans le chapitre suivant.

### 3.3 Présentation de RawCap :

RawCap est un sniffer réseau gratuit fonctionnant en ligne de commande et disponible pour Windows. Ce logiciel, qui ne nécessite pas d'installation, est disponible à l'adresse suivante : <http://www.netresec.com/?page=RawCap>. Voici ses caractéristiques :

- Capable de sniffer toutes les interfaces ayant une adresse IP, y compris 127.0.0.1 (localhost)
- RawCap.exe ne fait que 17 kB
- Aucune librairie externe ou DLL n'est nécessaire sauf .NET Framework 2.0
- Pas d'installation nécessaire juste lancer l'exécutable RawCap.exe pour sniffer
- Utilisation minimale de la mémoire et du CPU
- Sûr et facile à utiliser

Une fois le logiciel téléchargé, il suffit de double-cliquer dessus pour arriver à une interface une ligne de commande :

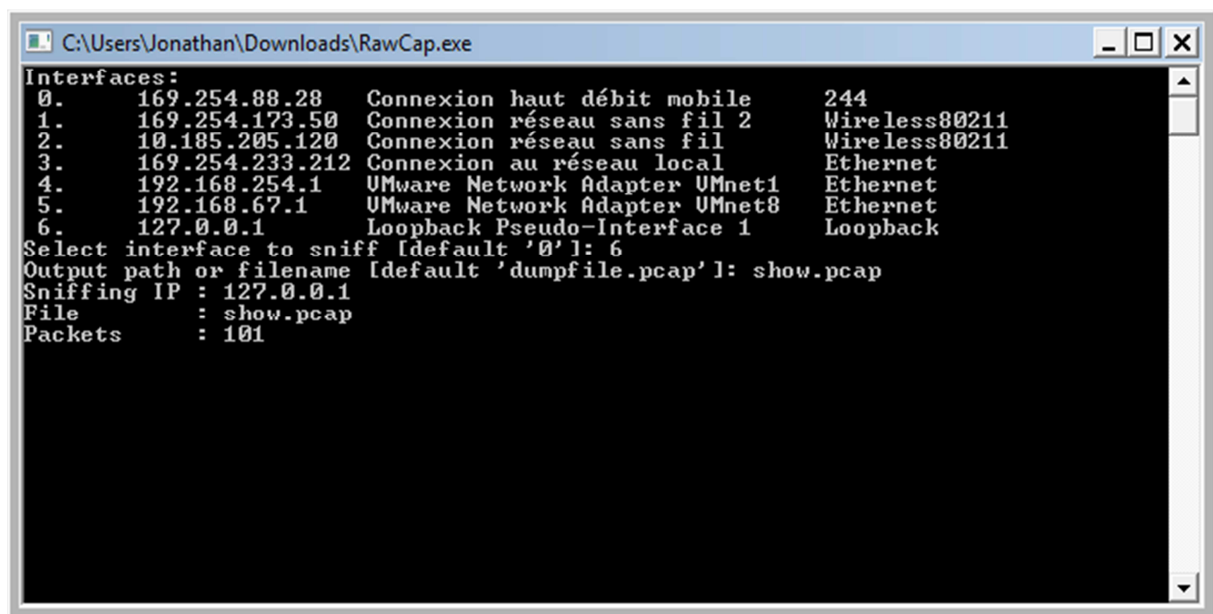


Figure 48 Logiciel RawCap

Ensuite, pour sniffer les paquets présents sur l'adresse 127.0.0.1, il suffit de sélectionner l'option 6 puis de donner un nom au fichier dans lequel les paquets vont être enregistrés. J'ai nommé ce fichier show.pcap afin que ce fichier soit compatible avec le logiciel Wireshark. Une fois que la capture des

paquets est terminée, le fichier peut être ouvert sans problème par le logiciel wireshark qui affichera les paquets capturés.

### 3.4 Analyse et envois des paquets à l'aide d'un serveur http :

Maintenant que le logiciel RawCap est installé, je fais une capture des paquets échangés lors de la connexion de la Wibox au Portal. Voici la description des paquets échangés entre ces deux entités, les premiers échanges sont des paquets http servant à ouvrir la connexion entre la Wibox et le Portal, c'est notamment lors de l'échange de ces premiers paquets que le login et le mot de passe sont échangés. Ensuite, c'est un paquet XML/http envoyé de la Wibox au Portal qui indique la version et le type d'encodage des prochains paquets XML. Une fois que la connexion a été établie, le Portal, va envoyer une requête afin d'indiquer à la Wibox de commencer à envoyer le flux de données provenant des capteurs (Get data). Pour commencer à envoyer le flux de données des capteurs, la Wibox va envoyer un paquet TCP à la Wibox en ouvrant le tag « Messages ». Une fois que ce tag a été ouvert, le flux de données peut commencer à transiter entre la Wibox et le Portal.

Le flux de données transitant entre la Wibox et le Portal se compose des paquets décrits dans la section « Types de paquets envoyés au Portal ».

### 3.5 Implémentation du serveur http :

Comme la librairie « Curl » que j'ai installée précédemment n'est faite que pour être un client http et que l'implémentation d'un serveur http serait très longue, je décide de prendre le code d'un serveur http déjà existant et disponible sur le net. J'ai trouvé le code d'un serveur http codé en C et relativement simple à l'adresse suivante : <http://blog.abhijeetr.com/2010/04/very-simple-http-server-written-in-c.html>. Il est possible de paramétrer le port d'écoute du serveur grâce à l'option -p, il est également possible de sélectionner son répertoire « Root » à l'aide de l'option -r. Par défaut, le port d'écoute du serveur sera le port 8088, comme sur la Wibox d'origine. Ce serveur peut gérer jusqu'à 1000 connexions en même temps.

La première chose que fait le serveur est tester les arguments reçus lors de son lancement. Ensuite, il va remplir avec des valeurs nulles son tableau de client. Ce tableau est un tableau d'entier et va servir stocker les différents file descriptor des sockets qui vont être assignées aux clients.

```
int i;
for (i=0; i<CONNMAX; i++)
clients[i]=-1;
startServer(PORT);

// ACCEPT connections
while (1)
{
    addrlen = sizeof(clientaddr);
    clients[slot] = accept (listenfd, (struct sockaddr *) &clientaddr, &addrlen);
```

Figure 49 Recupération du file descriptor d'un socket

Maintenant que le client a un file descriptor assigné, le programme va regarder si celui est correct et, si c'est le cas, va exécuter l'appel système fork(). Cet appel système, disponible sous les systèmes

Unix, permet de dupliquer un processus, cela va permettre aux clients d'avoir leurs propres espaces mémoire et de pouvoir s'exécuter en même temps.

Lorsque le processus a été dupliqué, une fonction nommée `respond()` est appelée et le file descriptor du client lui est passé en paramètre. La fonction `respond()` va permettre de communiquer avec le client http. C'est ici que les messages du client seront reçus et c'est aussi depuis cette fonction que le serveur va envoyer les différents messages notamment les paquets de `linkup`, `data` et `heartbeat`.

Par la suite, comme le serveur n'aura qu'une connexion avec le Portal, j'ai choisi de supprimer l'appel système `fork()` afin de ne pas dupliquer le processus. Ceci me permet de partager le même espace mémoire entre le serveur et le reste du programme et c'est très utile notamment pour partager de l'information entre ces deux entités grâce aux variables globales.

La première chose à faire lorsque le programme communique avec le Portal est d'envoyer un message de reset du statut de la Wibox. Ensuite, il faut attendre la requête « Get data » qui va être envoyée à la Wibox. Une fois que cette requête a été reçue, j'ai ouvert le tag XML « Messages » afin de pouvoir commencer à streamer les données des capteurs auprès du Portal. Une fois que ce tag est ouvert j'ai pu envoyer les différents paquets de données en les codants premièrement « en dur » afin de voir si le Portal réagissait bien et si mon faux capteur apparaissait comme étant connecté. J'ai donc créé une boucle `while` dans laquelle j'envoie les différents paquets afin de simuler le comportement d'un faux capteur. L'envoi des paquets est séparé par des pauses afin de ne pas surcharger la connexion.

```
// Envoi du flux de données au Portal.
while(1){
    send(clients[n], "5C\r\n<Health segno=\"0\" voltage=\"0\" originaddr=\"0\" sourc
    sleep(1);
    send(clients[n], "1A0\r\n<HaveData APChannel=\"1\" APMAC=\"00:23:69:41:75:80\" a
    sleep(1);
    send(clients[n], "85\r\n<Data mote_id=\"8869\" data_index=\"19390\" data_id=\"68
    sleep(1);
}
```

Figure 50 Envoi de fausses données au Portal

Les données que j'envoie dans cette boucle sont des données que j'ai recopiées de captures wireshark que j'ai effectuées lors de connexion réelles des capteurs avec le Portal. Afin de vérifier que ces données sont bien interprétées par le Portal, je me suis connecté au Portal comme expliqué dans la section suivante.

### 3.6 Connexion de la Wibox Lite au Portal :

Pour pouvoir connecter la Wibox au Portal, j'ai tout d'abord dû ajouter une nouvelle « Gateway » au Portal que j'ai nommée Wibox Lite. La Wibox complète conçue par Aginova et installée au début du projet fonctionne sur le même PC que le Portal c'est pour cela que son adresse est 127.0.0.1, mais pour la nouvelle Gateway, il est nécessaire de spécifier l'adresse IP et le port de celle-ci. Comme la Wibox Lite fonctionne sous ma machine virtuelle Linux, je récupère l'adresse IP de celle-ci grâce à la commande linux « `ifconfig` ».

**Create/edit a gateway**

*Make sure that the WiBox or RTLS you are trying to connect to is currently running !*

|                          |  |                          |  |
|--------------------------|--|--------------------------|--|
| Name* :                  | <input type="text" value="Wibox Lite"/>            | Enabled* :               | <input type="button" value="Yes"/>     |
| Host* :                  | <input type="text" value="10.185.205.243"/>        | Port* :                  | <input type="text" value="8088"/>      |
| Connect to* :            | <input type="button" value="WiBox / iBox Server"/> | Protocol* :              | <input type="button" value="Http"/>    |
| Web Services Username* : | <input type="text" value="ws"/>                    | Web Services Password* : | <input type="password" value="....."/> |

(\*) Required

Figure 51 Paramétrage de la Wibox Lite auprès du Portal

Maintenant que la Wibox Lite est créée, il faut désactiver le pare-feu Windows afin de laisser les paquets transiter à travers le pare-feu. On peut vérifier que la Wibox Lite se connecte sans problème au Portal en allant dans l'onglet « Administration » puis dans « Gateways » qui va nous montrer le statut des Wibox enregistrées auprès du Portal.

**Gateways administration**

*A gateway allows you to connect to a Wibox and/or a Real Time Location System (RTLS).*

| ID                         | Gateway name               | Enabled ? | URL  | Status    | Average uptime | Problems (if any) |
|----------------------------|----------------------------|-----------|--|-----------|----------------|-------------------|
| <input type="checkbox"/> 1 | <a href="#">localhost</a>  | Yes       | <a href="http://localhost:8088">http://localhost:8088</a> (IP: 127.0.0.1)                | Connected | 88.9%          | -                 |
| <input type="checkbox"/> 2 | <a href="#">Wibox Lite</a> | Yes       | <a href="http://10.185.205.243:8088">http://10.185.205.243:8088</a> (IP: 10.185.205.243) | Connected | 6.4%           | -                 |

Figure 52 Etat des Gateways reconnues par le Portal.

### 3.7 Disposition des capteurs sur le réseau:

Les capteurs Aginova vont pouvoir se connecter à la Wibox soit dans le même sous réseau ou alors derrière un NAT. On dit qu'un routeur fait du Network Address Translation(NAT) (« traduction d'adresse réseau ») lorsqu'il fait correspondre les adresses IP internes non-uniquees et souvent non routables d'un intranet à un ensemble d'adresses externes uniques et routables. Ce mécanisme permet notamment de faire correspondre une seule adresse externe publique visible sur Internet à toutes les adresses d'un réseau privé, et pallie ainsi l'épuisement des adresses IPv4.

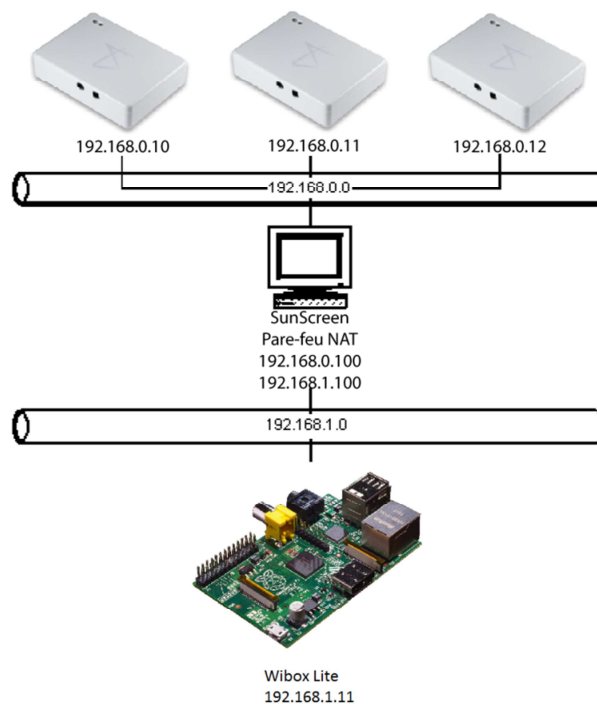


Figure 53 Schéma de disposition des capteurs

Lorsque les capteurs seront derrière un NAT, ils auront tous la même adresse IP et la seule façon de les différencier sera de regarder les ports sur lesquelles ils vont envoyer leurs données. Malheureusement, comme j'ai fini mon projet en Californie, je n'ai pas prévu de partir avec plusieurs capteurs, la société Aginova m'a donc envoyé des capteurs sur place, seulement ces capteurs étaient des vieux capteurs et ne pouvaient pas se connecter à la Wibox. Comme il n'était pas possible d'en envoyer des nouveaux à cause du manque de temps je n'ai pas pu implémenter la communication de la Wibox avec plusieurs capteurs.

J'ai par contre créé un algorithme qui permettrait la connexion de plusieurs capteurs à la Wibox lite si ceux-ci ont des adresses IP différentes. Par contre, je n'ai pas pu tester cet algorithme, vu que je n'ai qu'un seul capteur à disposition. Cet algorithme permet de gérer plusieurs sessions snmp avec différents capteurs mais pour que cela fonctionne totalement il faudrait aussi modifier quelque peu la réception des données afin d'acquiescer les paquets de chaque capteur ainsi que l'envoi des données au Portal ce qui n'a pas été implémenté.

Voici l'explication de cet algorithme. Après que les capteurs se soient authentifiés auprès de l'AP, ils vont envoyer des trappes SNMP afin de signaler à la Wibox qu'ils sont présents et qu'ils sont prêts à envoyer leurs données. C'est lors de la réception de ces trappes SNMP dans le thread trap\_listener que l'adresse du capteur envoyant les trappes va être réceptionnée. Pour lire l'adresse du capteur envoyant les trappes, j'ai utilisé la fonction inet\_ntoa(). Cette fonction retourne un string contenant l'adresse IP de la structure contenant cette adresse qui lui est passée en paramètre.

Afin que plusieurs capteurs puissent se connecter avec des adresses IP différentes à la Wibox Lite, j'ai déclaré un tableau qui va contenir les adresses IP des capteurs connectés à la Wibox. Premièrement, je mets l'adresse du capteur qui envoie la première trappe SNMP dans ce tableau, puis je crée un

thread de communication snmp auquel je vais passer l'adresse du capteur afin de pouvoir démarrer une session de communication avec celui-ci. Lorsque je reçois une trappe d'un nouveau capteur, je vais parcourir le tableau contenant les adresses des capteurs connectés et regarder si l'adresse du nouveau capteur est déjà présente dans ce tableau. Si l'adresse n'est pas encore présente, un nouveau thread va être démarré afin de commencer une nouvelle session SNMP avec l'adresse correspondante au nouveau capteur.

```
// Creation d'un nouveau thread si le capteur envoie sa premiere trame
if(index == 0 || match == 1){
    match = 0;
    memcpy(address_tab[index++], inet_ntoa(from.sin_addr), strlen(inet_ntoa(from.sin_addr)));
    char * test = malloc(16*sizeof(char));

    test = inet_ntoa(from.sin_addr);

    // Creation du thread snmp_listener
    if ((pthread_create(&SNMP, NULL ,snmp_listener, (void*)&test)) != 0) {
        printf("Thread snmp_listener non cree correctement");
        exit(1);
    }
}

// Test si l'adresse du capteur est déjà enregistree dans le tableau
match = 1;
for(i = 0; i < index; i++){
    if(strcmp(address_tab[i], inet_ntoa(from.sin_addr)) == 0){
        match = 0;
    }
}
```

Figure 54 Algorithme de création des threads SNMP

### 3.8 Découpage du programme en différents threads :

Afin d'avoir un programme le plus structuré et le plus efficace possible, je l'ai découpé en plusieurs threads POSIX. Les threads POSIX ou pthreads sont disponibles sur la plupart des systèmes UNIX modernes. Les différents threads sont créés et appelés dans le thread main() qui est le thread principal de ce programme. J'ai donc créé les threads à l'aide de la fonction pthread\_create() à laquelle il est nécessaire de passer en paramètre le thread créé ainsi que la fonction à laquelle ce thread est associé. Ensuite, j'ai utilisé la fonction pthread\_join() qui va attendre que l'exécution du thread passé en paramètre soit terminée, ce qui permet d'attendre la fin de l'exécution de tous les threads avant de terminer le programme.

```
// Creation du thread trap_listener
if ((err = pthread_create(&TRAP_LISTENER, NULL ,trap_listener, NULL)) != 0) {
    printf("Thread trap_listener non cree correctement");
    return EXIT_FAILURE;
}

// Creation du thread server_http
if ((err = pthread_create(&SERVER_HTTP, NULL ,server_http, NULL)) != 0) {
    printf("Thread server_http non cree correctement");
    return EXIT_FAILURE;
}

// Attente de la fin de l'execution des threads
pthread_join(DATA_LISTENER, NULL);
pthread_join(SNMP, NULL);
pthread_join(COMMAND_LISTENER, NULL);
pthread_join(TRAP_LISTENER, NULL);

return EXIT_SUCCESS;
```

Figure 55 Création de quelques threads et attente sur ceux-ci

Je vais résumer ici les différents threads que j'ai créés pour le programme de la Wibox Lite et expliquer le but de ceux-ci :

### 3.8.1 Data\_listener :

Le thread data\_listener va ouvrir une communication UDP grâce à un socket écoutant sur le port de réception des données envoyées par le capteur. C'est dans ce thread que les numéros des données vont être extraits afin de pouvoir être acquittés auprès capteur et c'est aussi ce thread qui va insérer les données dans la base de données SQLite.

### 3.8.2 Trap\_listener :

Ce thread va ouvrir une communication snmp afin de récupérer les trappes snmp qui sont envoyées du capteur à la Wibox. Différents types de trappes sont envoyées par les capteurs. Il peut y avoir soit des messages de reconfiguration « Configuration update » qui signifient que le capteur est prêt à recevoir des commandes de reconfiguration ou alors des trappes de « Heartbeat » qui contiennent des informations propres au capteur et qui doivent être insérées dans la base de données. Ce thread s'occupe de réagir de fonction adéquate suivant le type de trappe reçue.

### 3.8.3 Command\_listener :

Ce thread va appeler une fonction qui va permettre de communiquer avec le programme d'envoi des commandes au server (server\_tcp.c). Autrement dit, cette fonction va écouter les informations reçues sur le socket établi avec le programme d'envoi des données afin de réceptionner les commandes à envoyer au capteur à reconfigurer. Il se peut que plusieurs commandes soient envoyées au même capteur, car l'utilisateur veut reconfigurer plusieurs paramètres de ce capteurs dans ce cas les commandes seront stockées dans une liste chaînée de commandes qui vont ensuite être envoyées au capteur par le thread snmp\_listener.



### 3.8.4 Snmp\_listener :

Ce thread va tout d'abord envoyer des requêtes snmp afin d'extraire les informations du capteur. Ensuite, il va se mettre en attente sur un mutex jusqu'à ce qu'une trappes snmp soit reçues par le thread trap\_listener. Lorsqu'une trappe de configuration est reçue, le thread trap\_listener va relâcher le mutex ce qui va permettre au thread snmp\_listener de continuer son exécution, il va alors pouvoir envoyer les messages de reconfigurations aux capteurs ou encore acquitter les paquets de données envoyés par le capteur. L'attente sur ce mutex est nécessaire, car il n'est possible de reconfigurer les capteurs que lorsqu'une trappe de configuration est reçue, ceci est fait pour la faible consommation des capteurs.

### 3.8.5 Server\_http :

Ce thread va lancer le serveur http qui va permettre la communication avec le Portal. Le serveur http va se connecter au Portal puis envoyer les informations transmises par les capteurs afin de pouvoir les afficher à l'utilisateur final. Le serveur peut traiter jusqu'à 1000 connexions en même temps. Il va envoyer différentes informations au Portal comme des paquets de « Heartbeat », des paquets de données et encore des paquets servant à maintenir la connexion entre le server et le Portal.

## 3.9 Envoi des données au Portal :

Comme je l'ai expliqué dans le chapitre « Récupérations des paquets Heartbeat », je récupère toutes les informations contenues dans les paquets de heartbeat afin de pouvoir les transmettre au Portal. Je récupère aussi les paquets de données afin de pouvoir les parser et envoyer leur charge utile au Portal.

Les paquets qui sont envoyés au Portal sont des paquets http qui sont créés d'une manière bien précise, on ne peut pas seulement envoyer les informations contenues dans les paquets UDP envoyés par le capteur. Pour formater correctement ces paquets, j'ai analysé les paquets envoyés par la Wibox installée sur mon ordinateur au Portal installé lui aussi sur mon ordinateur. Afin de tester si le capteur apparaissait comme connecté sur la Wibox, j'ai remplacé les informations que ces paquets contenaient par les informations réelles envoyées par les capteurs.

Pour remplacer ces informations, j'ai utilisé la fonction `sprintf()` qui permet de formater un string avec les données désirées. Pour les paquets de Linkup, la fonction `sprintf()` n'a pas été nécessaire, car ce sont toujours les mêmes paquets qui vont être envoyés au Portal. Par contre, pour les paquets de données ainsi que pour les paquets de heartbeat, les données vont bien sûr être différentes pour chaque paquet échangés.

Comme la fonction `sprintf()` ajoute le caractère '\0' en fin de ligne, je peux utiliser la fonction `strlen()` qui va retourner la longueur de la chaîne de caractère passée en paramètre afin de connaître la taille de la chaîne de caractère qui va être transmise au socket.

Voici les informations qui sont envoyées dans les paquets de données :

- Mote\_id : Numéro du capteur
- Data\_index : Numéro du dernier paquet de données acquitté par le capteur
- Data\_id : Numéro du dernier paquet de données reçu par le capteur
- Timestamp : Valeur représentant la date et l'heure des données



- Data\_type : Type des données (peut changer suivant le capteur utilisé)
- Data : Données mesurées par le capteur

J'ai extrait toutes ces informations des paquets de données qui sont transmis par UDP sur le port de 163. Je n'ai par contre pas pu trouver l'information sur le champ data\_type j'ai donc laissé la valeur de ce champ à 13 ce qui correspond au type de données envoyée par le capteur Sentinel PRO II.

Voici les informations qui sont envoyées dans les paquets de heartbeat :

- APChannel : Numéro du canal utilisé par le réseau WiFi.
- APMAC : Adresse MAC de l'AccessPoint.
- AssPrd : Valeur par défaut : 900
- CodeVersion : Version du logiciel installé sur le capteur
- ComMode : Valeur par défaut : 0
- Count : Nombre de paquets de données stocké par le capteur
- Gr1 : Temps de la dernière modification du groupe 1
- Grp1 : Temps de la dernière modification du groupe 1 (Portal)
- Gr2 : Temps de la dernière modification du groupe 2
- Grp2 : Temps de la dernière modification du groupe 2 (Portal)
- Gr3 : Temps de la dernière modification du groupe 3
- Grp3 : Temps de la dernière modification du groupe 3 (Portal)
- Gr4 : Temps de la dernière modification du groupe 4
- Grp4 : Temps de la dernière modification du groupe 4 (Portal)
- IsReader : Valeur par défaut : false
- Mac : Adresse mac du capteur
- Mote\_id : Numéro du capteur
- Parent : Valeur par défaut : 0
- Product : Numéro de produit
- Resets : Nombre de resets depuis que le capteur a été flashé.
- RetryCount : Valeur par défaut : 0.037
- RSSI : Force du signal (en dBm).
- StoreFwdEnabled : Indique si le capteur va stocker les paquets de données ou non.
- Time : Temps d'envoi du paquet (nombre de secondes depuis le premier janvier 2006)
- UpTime : Temps depuis lequel le capteur est actif.
- Voltage : Voltage de la batterie du capteur.

Comme pour l'envoi des paquets de données, j'ai récupéré la majorité de ces informations dans les paquets de heartbeat envoyés par le capteur. Je n'ai par contre pas pu extraire différentes informations dont j'ai laissé leur valeur par défaut comme vous pouvez le voir dans la liste ci-dessus.

Pour finir, le dernier type de paquet envoyé est un paquet de Linkup qui sert simplement à garder la connexion entre le capteur et l'AccessPoint en vie en envoyant des paquets séparés par des intervalles de temps. Ces paquets sont toujours les mêmes et font donc toujours la même taille, c'est

pourquoi je les envoie seulement à l'aide de la fonction `send()` en lui passant une taille fixe de caractère. Voici ce que ces paquets contiennent :

- Seqno : Valeur fixée à 0.
- Voltage : Valeur fixée à 0.
- Originaddr : Valeur fixée à 0.
- Sourceaddr : Valeur fixée à 0.
- Quality : Valeur fixée à 0.
- RSSI : Valeur fixée à 0.

Comme les données contenues dans ce paquet ne sont pas utilisées, elles ont toutes été fixées à zéro, ce paquet servant juste à garder la connexion « alive ».

Pour envoyer les paquets de données ainsi que les paquets de heartbeat, j'ai créé un thread nommé `data_send` auquel je passe le file descriptor du socket en argument. Ce thread est créé par la fonction `respond()` après que celle-ci ait envoyé les messages initiant la connexion avec le Portal. Le thread `data_send` va attendre sur un mutex qu'une nouvelle donnée soit arrivée, ce mutex va être relâché par le thread `data_listener` chaque fois qu'un nouveau paquet de donnée est reçu par la Wibox.

Il est également nécessaire de contrôler que les paquets que l'on envoie n'aient pas déjà été envoyés. Pour ce faire, je sauvegarde l'ID du paquet envoyé précédemment et je n'envoie le nouveau paquet que si son ID est supérieur à l'ID précédent.

### 3.10 Consommation mémoire et CPU du programme de la Wibox Lite :

Une des choses importantes à examiner est la consommation mémoire ainsi que la consommation du processeur. Comme ce programme va être exécuté sur le Raspberry Pi, j'ai fait ces mesures sur ce dernier lorsque le programme était en cours d'exécution. J'ai utilisé la commande `ps` qui dresse une liste des processus actifs à laquelle j'ai passé le nom de mon application à l'aide de l'argument `-C`. Ensuite, il suffit de sélectionner les paramètres `pmem` et `pcpu` qui vont nous indiquer la consommation mémoire ainsi que la consommation cpu de l'application.

Lors de mes tests sur le Raspberry Pi, j'ai constaté que mon application ne consommait que 1% de la mémoire vive c'est-à-dire environ 2,5 Mo et qu'elle consommait seulement 0.1% des ressources du processeur.

### 3.11 Correction de bugs et amélioration du programme :

J'ai réalisé plusieurs tests afin de contrôler le fonctionnement de mon programme et j'ai effectué plusieurs contrôles sur toutes les saisies utilisateur afin de rendre mon programme le plus robuste possible. Lors de ces tests, j'ai tout d'abord remarqué que lorsque je me connectais à la Wibox Lite avec le programme de reconfiguration des capteurs, je ne pouvais pas quitter ce programme sans faire planter le programme de la Wibox. Ceci vient du fait que, comme la communication socket qui est utilisée pour la communication entre ces deux programmes utilise le protocole TCP, je suis obligé de fermer la connexion dans les deux sens.

Pour ce faire, j'ai créé une nouvelle commande, disponible dans le menu du programme de reconfiguration des capteurs, qui permet de quitter le programme proprement et sans causer aucun

problème au programme de la Wibox Lite. Cette commande ferme tout simplement le socket et envoie une commande au programme de la Wibox Lite afin de lui indiquer qu'il faut aussi qu'il ferme la communication socket de son côté.

Afin que l'utilisateur ne puisse pas quitter le programme autrement qu'en utilisant cette commande, j'ai récupéré le signal CTRL+C qui normalement termine le programme et j'ai ajouté, un texte qui dit que pour bien terminer le programme il faut utiliser la commande prévue à cet effet. Ceci améliore grandement la robustesse du programme.

Ensuite au début de la conception du programme, j'ai créé un thread nommé `command_listener`. Ce thread s'occupait de la recevoir les commandes envoyées par le programme de reconfiguration des capteurs. Le problème est que lorsque le programme de reconfiguration des capteurs se terminait, le thread `command_listener` présent dans la Wibox se terminait lui aussi et ainsi on ne pouvait lancer une nouvelle fois le programme de reconfiguration des capteurs.

Pour pallier à ce problème, j'ai copié les opérations effectuées dans le thread `command_listener` à l'intérieur d'une fonction nommée `command_receiver()`. Lorsque le thread `command_listener` est créé il va donc appeler cette fonction qui va s'occuper de la communication avec les capteurs. Lorsque cette fonction se termine, c'est-à-dire lorsque le programme de reconfiguration des capteurs est terminé, la fonction va être rappelée par le thread `command_listener` qui tourne en boucle. Cette façon de faire permet de pouvoir se reconnecter indéfiniment avec le programme de reconfiguration des capteurs à la Wibox Lite.

Une dernière chose que j'ai remarquée dans le programme est que, lorsque je changeais le port sur lequel le capteur doit envoyer les données, je n'étais plus capable recevoir ces dernières car je ne changeais jamais le port sur lequel j'écoutais ces données. Pour résoudre ce problème, j'ai défini une variable globale nommée `dataPort` que j'initialise à zéro. Lorsque j'envoie la commande changeant le port de données, le programme va mettre le nouveau numéro de port dans cette variable. Dans le thread `data_listener`, si cette variable n'est pas à zéro, le socket qui par défaut écoute sur le port 163 va être réinitialisé pour écouter sur le nouveau port de données. Je peux maintenant changer le port de destination des données sans pour autant compromettre le fonctionnement de l'application.

```
// Test si les donnees sont envoyees sur un autre port
if(dataPort != 0){
    // Si c'est le cas on ferme la socket
    close(sock);
    socket(AF_INET, SOCK_DGRAM, 0);
    setsockopt(sock, SOL_SOCKET, SO_REUSEADDR, &so_reuseaddr, sizeof so_reuseaddr);
    memset(&sin, 0, sizeof(struct sockaddr_in));
    sin.sin_addr.s_addr = htonl(INADDR_ANY);
    sin.sin_family = AF_INET;

    // Ecoute sur le nouveau port de donnees
    sin.sin_port = htons(dataPort);
    // Bind de la socket
    if(bind(sock, (SOCKADDR *) &sin, sizeof sin) == SOCKET_ERROR)
    {
        perror("bind()");
        exit(1);
    }
    dataPort = 0;
}
```

Figure 56 Changement du port d'écoute de la socket

### 3.12 Plateforme embarquée Raspberry Pi

La version lite de la Wibox que je suis en train de développer va devoir être installée sur une plateforme embarquée standard tournant sur Linux. Il existe de nombreuses cartes embarquées fonctionnant avec différentes distribution Linux mais pour ce projet, nous avons choisi d'utiliser une carte nommée Raspberry pi qui est devenu un standard dans le monde de l'informatique embarquée.



Figure 57 Raspberry Pi

Le Raspberry pi est un ordinateur miniature embarqué qui a le format d'une carte de crédit. Ce périphérique est notamment utilisé pour l'apprentissage de la programmation informatique. Ses caractéristiques sont les suivantes :

- Processeur ARM1176JZF-S (ARMv6) cadencé à 700 MHz. Ce type de processeur est un processeur basse consommation et est similaire aux processeurs utilisés dans la majorité des smartphones.
- 256 Mo de mémoire RAM
- 2 Sorties vidéo : Composite et HDMI
- 1 Sortie audio stéréo Jack 3,5 mm (sortie 5.1 sur la prise HDMI)
- Unité de lecture-écriture de carte mémoire : SDHC / MMC / SDIO
- 2 Port USB 2.0
- Prise pour alimentation Micro-USB (consommation : 400mA + périphériques)
- Des entrées / sorties supplémentaires devraient être accessibles directement sur la carte mère via des pins 3v3 (à confirmer : GPIO, S2C, SPI)

- API logicielle video : OpenGL : version embarquée OpenGL ES 2.0
- Décodage vidéo : 1080p30 H.264 high-profile
- 1 port réseau Fast Ethernet (10/100 Mbits/s)

Cette plateforme est bon marché, car elle est destinée à un très large public de développeur. Par contre, il est nécessaire d'avoir un équipement supplémentaire afin de pouvoir utiliser toutes les fonctionnalités offertes par le Raspberry pi. Cet équipement est le suivant :

- Un écran ou un téléviseur qui peuvent être connectés par HDMI ou VGA directement au périphérique.
- Un clavier et une souris qui peuvent être branchés aux ports USB.
- Une carte mémoire SD qui va contenir le système d'exploitation de la carte.
- Un boîtier.
- Une alimentation. Heureusement, le Raspberry Pi est compatible avec de nombreuses alimentations micro-USB qui sont notamment utilisées sur les smartphones. Cette alimentation doit avoir une tension de 5V et doit pouvoir fournir un courant de 700 mA.

Comme la version de la Wibox Lite doit se connecter aux différents capteurs via un point d'accès, il a été nécessaire de commander un dongle Wifi compatible avec le Raspberry Pi. Pour ce faire j'ai commandé le composant Edimax EW-7811UN qui dispose de l'USB 2.0 et des normes Wifi 802.11 b/g/n. J'ai aussi commandé un adaptateur HDMI vers DVI afin de pouvoir connecter la sortie vidéo aux écrans disponibles dans les bureaux Aginova.

### 3.13 Démarrage du Raspberry Pi :

Pour pouvoir me connecter la première fois à la plateforme Raspberry Pi, j'ai dû effectuer les opérations suivantes :

Premièrement, j'ai téléchargé l'OS nommé Raspbian et disponible à l'adresse suivante : <http://downloads.raspberrypi.org/images/raspbian/2012-12-16-wheezy-raspbian/2012-12-16-wheezy-raspbian.zip>. Raspbian est un système d'exploitation gratuit basé sur Debian et spécialement optimisé pour le hardware du Raspberry Pi. Ce système d'exploitation comprend les programmes basiques ainsi que les différents utilitaires nécessaires au bon fonctionnement du Raspberry Pi. Il faut ensuite extraire le fichier .img contenu du zip téléchargé.

Une fois le fichier image obtenu, il est nécessaire de télécharger le programme Win32DiskImager software afin de copier le fichier image sur la carte SD qui va être insérée dans le Raspberry Pi. Cet utilitaire est également gratuit et disponible à l'adresse suivante : <https://launchpad.net/win32-image-writer/+download>.

Lorsque la carte mémoire a été écrite, on peut l'insérer dans le Raspberry Pi afin d'effectuer son premier démarrage. Lors du premier boot, on arrive dans la fenêtre de configuration du Raspberry Pi. Dans cette fenêtre, nous pouvons changer les paramètres de location ainsi que le fuseau horaire utilisé. Pour finir il faut sélectionner la seconde option `expand_rootfs` et rebooter l'appareil.

Le login par défaut du Raspberry Pi est « pi » tandis que le mot de passe est « raspberry ». Nous pouvons maintenant nous connecter à l'appareil, ouvrir un shell et tapé la commande startx afin de démarrer l'interface graphique :

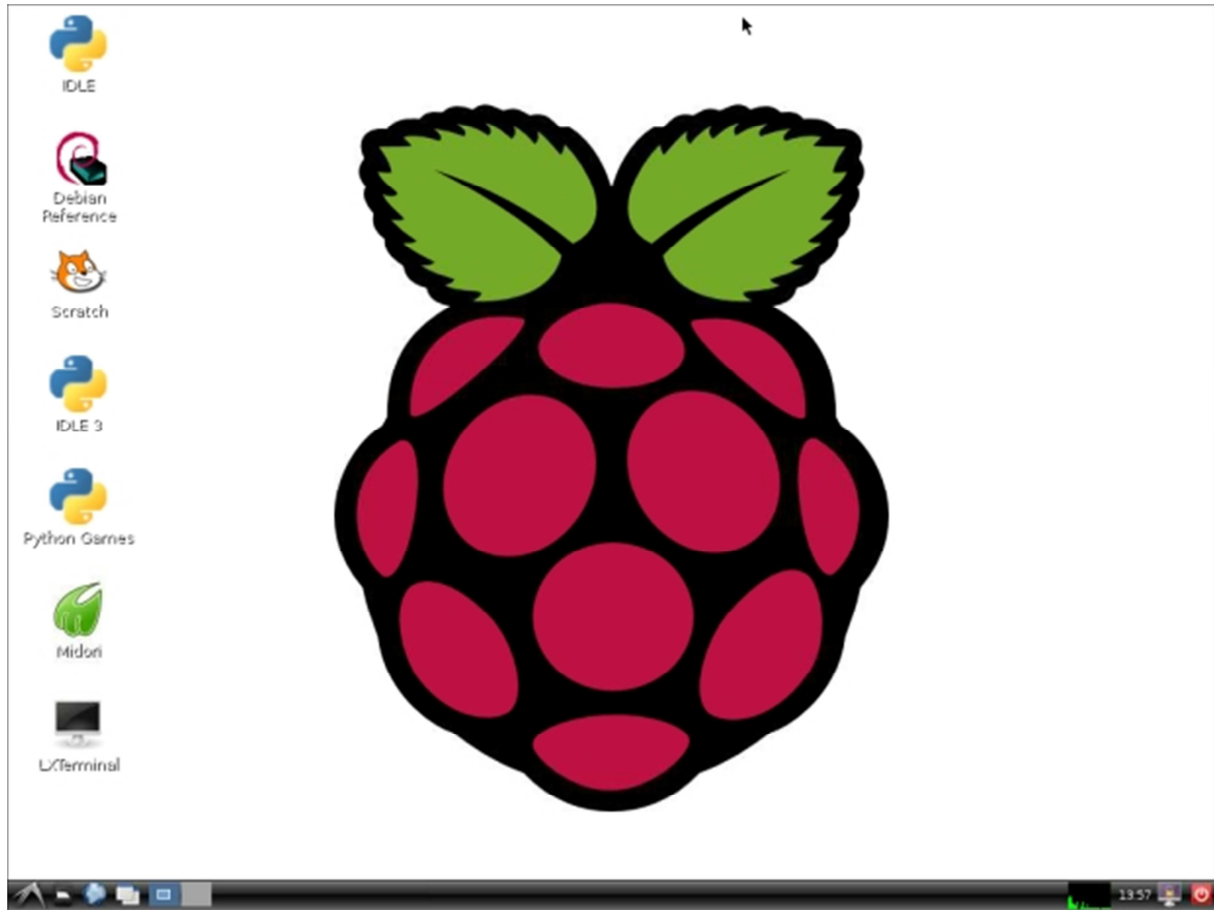


Figure 58 Desktop du Raspberry Pi

L'interface graphique du Raspberry Pi est surtout très utile pour configurer le réseau Wifi que l'on désire rejoindre. Une fois que le réseau Wifi ou câblé est rejoint et que mon ordinateur portable est sur ce même réseau, je peux sans autre transférer des fichiers de mon ordinateur au Raspberry pi en utilisant la commande `sudo scp <nom du fichier> pi@<adresse>:/home/pi/Desktop`. Il est également possible de se connecter en SSH au Raspberry pi en utilisant la commande `ssh -l pi <adresse>`. Il faudra bien sûr entrer le mot de passe de celui-ci afin de se connecter dessus.

Une fois que nous sommes connectés sur le Raspberry Pi en ligne de commande, nous disposons d'une interface en ligne de commande tout à fait standard. Il est possible de compiler des programmes directement sur le Raspberry pi grâce au compilateur gcc qui est installé d'office sur la distribution Raspbian.



### 3.14 Configuration du réseau sans fil sur le Raspberry Pi :

Pour configurer l'interface Wifi du Raspberry Pi, il est nécessaire d'éditer le fichier de configuration du réseau. Ce fichier est présent sur chaque système Linux et se trouve à l'emplacement suivant : `/etc/network`. Le fichier à éditer se nomme `interfaces`. Ce fichier contient le paramétrage réseau de chaque interface du Raspberry Pi. Premièrement il y a l'interface de Loopback `Lo`, ensuite il y a bien sur l'interface ethernet nommée `eth0`, puis j'ai rajouté l'interface Wifi nommée `wlan0`.

Voici la configuration que j'ai faite dans ce fichier pour que l'interface Wifi `wlan0` se connecte automatiquement au réseau Aginova :

- `allow-hotplug wlan0` : Sert à détecter l'interface wifi qui sera branchée en USB
- `auto wlan0` : Cela signifie que l'interface sera démarrée automatiquement au démarrage du Raspberry Pi.
- `iface wlan0 inet dhcp` : Signifie que cette interface est un client DHCP, ce qui signifie que le point d'accès va donner une adresse dynamiquement à cette interface.
- `wpa-ssid « aginova »` : Comme son nom l'indique, cette ligne configure nom du réseau sans fil auquel se connecter
- `wpa-psk « aginova1234 »` : Cette ligne contient la passphrase du réseau sans fil sécurisé à rejoindre.

Maintenant que la configuration du réseau sans fil est terminée, le Raspberry Pi se connectera sans problème à ce réseau lors de chaque démarrage si celui-ci est disponible.

### 3.15 Lancement du programme au démarrage du Raspberry Pi :

Afin de ne pas avoir besoin de se connecter au Raspberry et de devoir chaque fois exécuter le programme de la `Wibox_Lite` pour le lancer, j'ai trouvé qu'il serait intéressant que le programme se lance automatiquement au démarrage de la plateforme. Pour ce faire, j'ai créé un script d'initialisation qui sera appelé au démarrage du Raspberry Pi. Ce script se nomme `Wibox_lite.sh` et va être placé à l'emplacement suivant : `/etc/init.d`

Ce script doit être exécutable, c'est pourquoi il faut exécuter la commande `chmod 777 Wibox_lite.sh`.

Maintenant que le script est créé, il faut remplir les actions que celui-ci va devoir exécuter. Lorsque le service est démarré, il faut lancer le programme en ligne de commande, j'ai donc rentré les deux commandes ci-dessous afin de lancer le programme :

- `cd /bin/`
- `sudo ionice -c 2 -n 0 nice -n -20 ./Wibox_lite`

La première ligne sert à se déplacer dans le répertoire contenant l'exécutable du programme de la `Wibox` tandis que la deuxième ligne sert à exécuter le programme avec la plus haute priorité possible afin que le processeur le préempte le moins possible.

Pour tester le script il suffit simplement de l'exécuter et de regarder dans les processus en cours si l'application est en marche ce qui est le cas. Maintenant que l'on sait que le script fonctionne, il faut que ce script soit exécuté au démarrage de l'application. Pour ce faire, il faut exécuter la commande

update-rc.d Wibox\_lite.sh defaults. Cette commande insère l'exécution du script Wibox\_lite.sh lors de la séquence de démarrage de l'appareil.

Une fois que cette dernière commande a été entrée, le programme se lance sans problème lors de chaque démarrage du Raspberry Pi, il est à noter que l'on peut facilement enlever le programme du script de démarrage en exécutant simplement la commande `update-rc.d Wibox_lite.sh remove`.

Comme le Raspberry pi a beaucoup moins de puissance que mon ordinateur portable, je décide de faire de la compilation croisée c'est-à-dire que je vais compiler un programme pour le Raspberry pi sur mon ordinateur portable afin de gagner du temps lors de la compilation. Ensuite, il ne me restera plus qu'à transférer le fichier exécutable sur le Raspberry pi afin de pouvoir lancer le programme. Le chapitre suivant explique la mise en place de l'outil de cross-compilation que j'ai installé sur mon ordinateur portable.

### 3.16 Installation de la toolchain pour le Raspberry Pi :

Pour installer les bons utilitaires nécessaires à la cross-compilation, j'ai suivi le tutoriel présent à l'adresse suivante :

[http://www.chicoree.fr/w/Compilation\\_crois%C3%A9e\\_facile\\_pour\\_Raspberry\\_Pi#Installer\\_crosstool-ng](http://www.chicoree.fr/w/Compilation_crois%C3%A9e_facile_pour_Raspberry_Pi#Installer_crosstool-ng). Ce tutoriel explique comment faire l'installation de la toolchain qui va nous permettre de cross-compiler un programme depuis ma machine pour une architecture ARM. Premièrement, il est nécessaire d'installer les paquets suivants sur ma machine Linux :

- `apt-get install bzip2`
- `apt-get install build-essential`
- `apt-get install bison`
- `apt-get install flex`
- `apt-get install gperf`
- `apt-get install texinfo`
- `apt-get install gawk`
- `apt-get install libtool`
- `apt-get install automake`
- `apt-get install libnurses5-dev`
- `apt-get install subversion`

Ces paquets sont nécessaires à l'installation de l'utilitaire crosstool-NG. Crosstool-NG est un utilitaire qui va nous permettre de fabriquer sur mesure les outils permettant de faire de la compilation croisée. Une fois les paquets mentionnés précédemment sont installés, on peut sans autre télécharger crosstool-NG disponible à l'adresse suivante : <http://crosstool-ng.org/download/crosstool-ng/crosstool-ng-1.18.0.tar.bz2>. Maintenant que nous avons téléchargé l'utilitaire, il ne nous reste plus qu'à le décompresser et à l'installer à l'aide des commandes suivantes :

- `tar -xjf crosstool-ng-<num de version>`
- `cd crosstool-ng-<num de version>`
- `./configure --prefix=/opt/crosstool-ng`



- make
- sudo make install

La version de crosstool-ng utilisée pour mon projet est la version 1.18.0 qui est la plus récente à l'heure où j'écris ce rapport. Un échec lors de l'exécution du script « configure » signifie souvent qu'il manque un pré-requis à la machine Linux. Pour parer à ce problème, il faut vérifier que tous les prérequis soient bien installés et si ce n'est pas le cas essayer de localiser et d'installer manuellement les logiciels manquants.

Cette installation a seulement installé les scripts nécessaires pour fabriquer les outils de compilation croisée. Il faut encore configurer crosstool-NG afin de lui indiquer l'architecture de la cible, la version du kernel ou encore la version du compilateur qui nous intéresse. Les options de configuration sont très nombreuses c'est pourquoi je vais faire un explicatif de la configuration à entrer pour créer une toolchain compatible avec le Raspberry pi.

Premièrement, pour configurer crosstool-ng, il est nécessaire d'entrer les lignes de commandes suivantes dans Linux :

- export PATH="{PATH}:/opt/crosstool-ng/bin"
- mkdir my-dev
- cd my-dev
- ct-nt menuconfig

Menuconfig est une interface permettant de modifier un certain nombre d'options afin de créer une toolchain ARM pour le Raspberry Pi. Voici la configuration de crosstool-ng que j'ai effectuée :

Dans le menu « Paths and misc options » cocher « Try feature marked as experimental », cela permet en particulier de choisir le compilateur « linaro » lors d'une étape ultérieure. Une fois cette modification effectuée, revenir au menu principal en sélectionnant « Exit ».

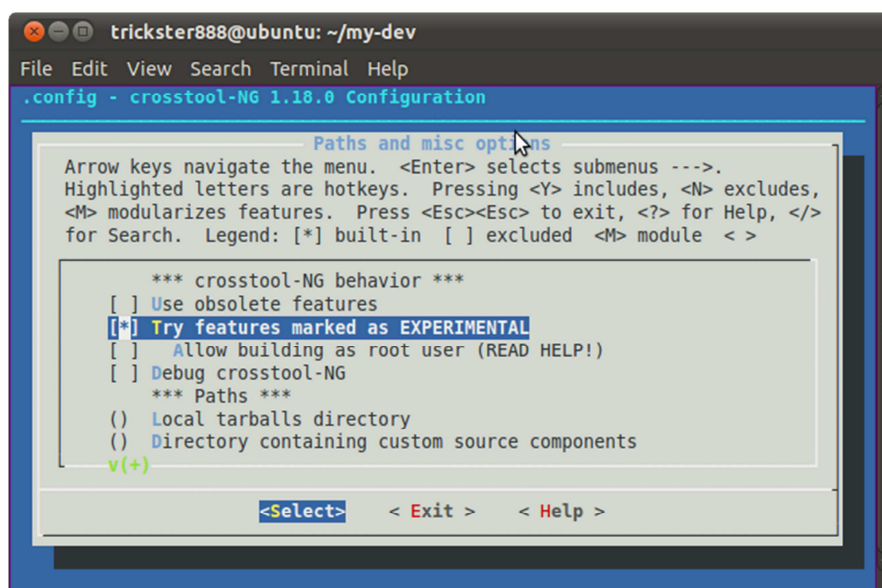


Figure 59 Menu « Paths and misc options »

Dans le menu « Target options », pour « Target Architecture » choisir ARM, c'est le type d'architecture du processeur utilisé par le Raspberry. Ensuite, pour l'option « Endianness » il faut vérifier que little endian est bien sélectionné et s'assurer que « bitness » est bien sur 32 bits. Puis revenir au menu principal en sélectionnant « Exit ».

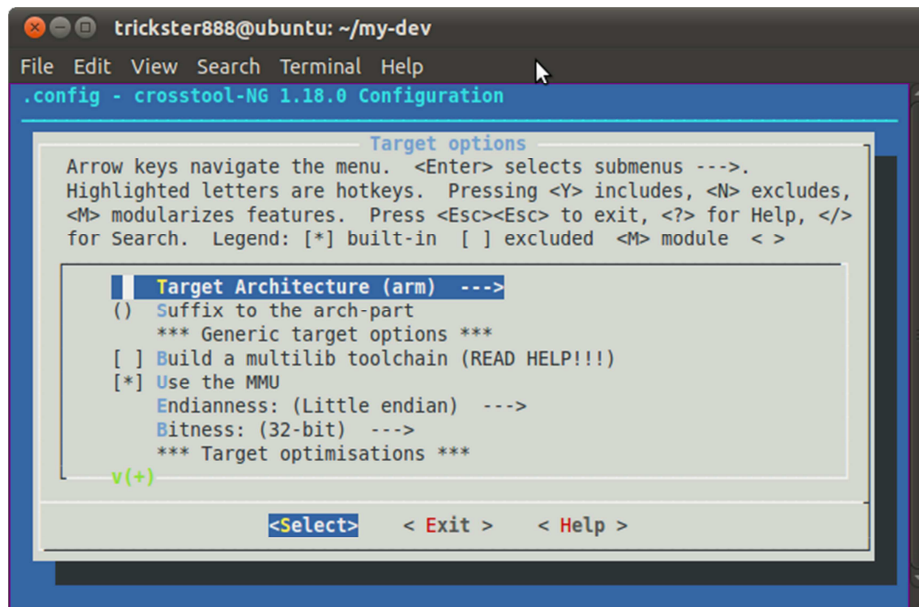


Figure 60 Menu « Target options »

Dans le menu « Operating System », choisir Linux comme système d'exploitation.

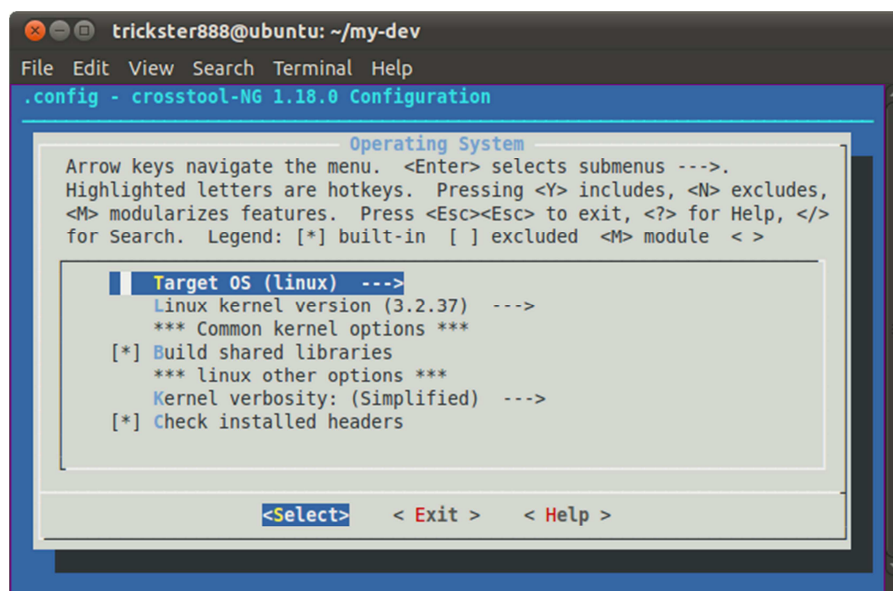


Figure 61 Menu « Operating System »

Ensuite, dans le menu « Binary utilities », choisir la version la plus récente possible de binutils en évitant si possible les versions expérimentales. Puis revenir au menu principal en sélectionnant « Exit ».

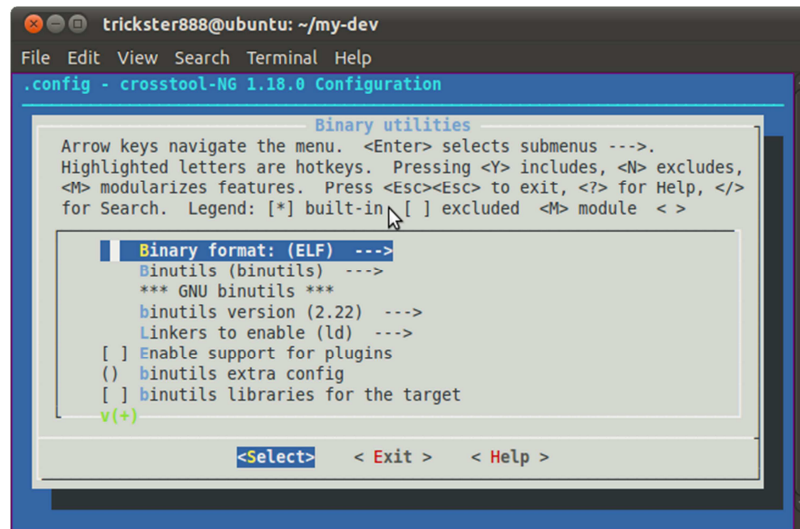


Figure 62 Menu « Binary utilities »

Puis dans le menu « C compiler », il est nécessaire d'activer « Show Linaro versions (EXPERIMENTAL) » afin de pouvoir sélectionner le compilateur gcc. Dans « gcc version », choisir un compilateur récent. Ensuite revenir au menu principal en sélectionnant « Exit ».

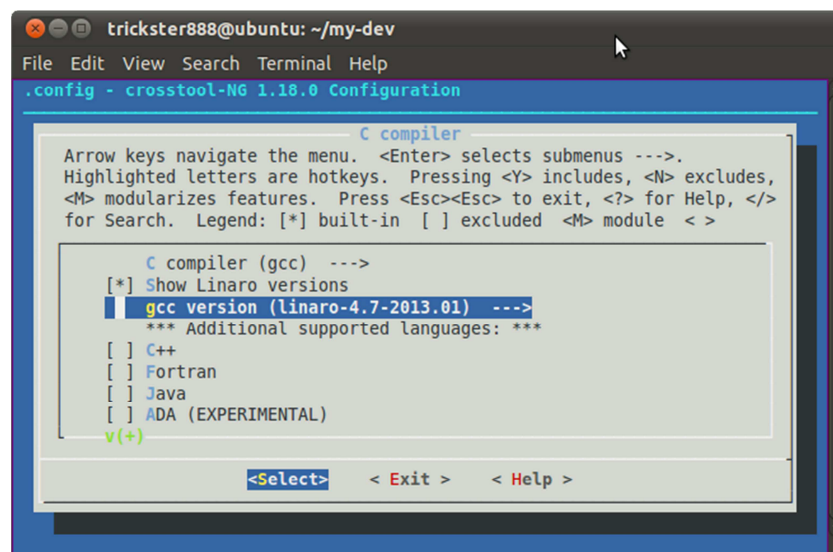


Figure 63 Menu « C compiler »

Nous avons maintenant fini la configuration de la toolchain pour le Raspberry pi. Pour sauvegarder les configurations que nous venons de mettre en place, il nous reste plus qu'à sélectionner « Exit » et à choisir de sauvegarder les modifications effectuées.

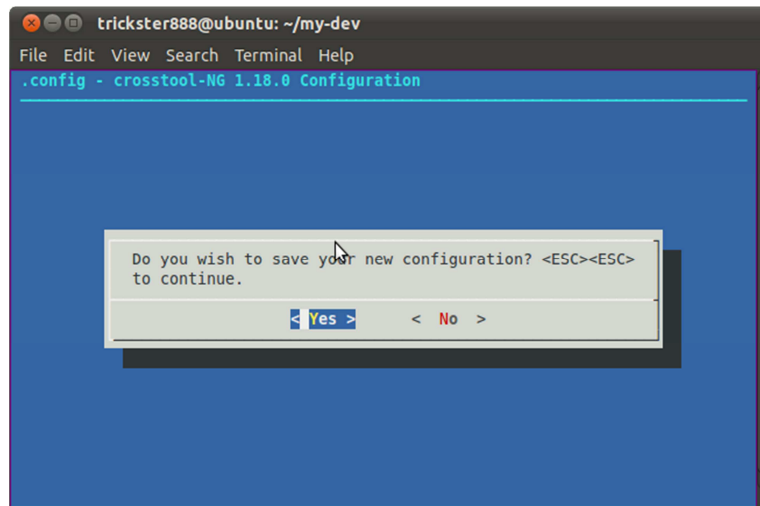


Figure 64 Sauvegarde de la configuration

Maintenant que la configuration est terminée, on peut sans autre lancer la compilation et l'installation de la toolchain à l'aide de la commande :

- `ct-ng build`

La compilation et l'installation de la toolchain est une opération relativement lourde c'est pourquoi elle peut prendre beaucoup du temps (dépendamment du pc utilisé cela varie entre 20 minutes et 1h30).

Pour pouvoir utiliser cette toolchain, il faut encore ajouter le chemin vers les outils de la toolchain au path grace à la commande suivante :

- `export PATH="${PATH}:/home/jonathan/x-tools/arm-unknown-linux-gnueabi/bin"`

Maintenant que ceci est fait il ne faut plus utiliser `gcc` pour compiler un programme mais il faut utiliser `arm-unknown-linux-gnueabi-gcc`. Une fois que cette étape est terminée, la toolchain est fonctionnelle et peut être utilisée afin de compiler des programmes sur le Raspberry pi.

## 4 Mise en place de la Wibox Lite :

Ce chapitre vise à décrire comment mettre en place de A à Z la Wibox Lite sur une plateforme embarquée, dans mon exemple un Raspberry Pi, et comment configurer les différents paramètres nécessaires à son bon fonctionnement.

Tout d'abord, il faut avoir le matériel suivant à disposition :

- Ordinateur avec le Portal d'Aginova installé
- Une plateforme embarquée Linux comportant une interface Wifi ainsi que les librairies, Net-SNMP, SQLite et Curl installée
- Un point d'accès Wifi
- Le logiciel de la Wibox Lite ainsi que les différents scripts nécessaire à disposition
- Un capteur Aginova de type Sentinel PRO II

Pour commencer, il est nécessaire de configurer le point d'accès Wifi. Lors du premier démarrage des capteurs ou lors du reset de ceux-ci, ils sont configurés pour se connecter à un réseau comportant le ssid « aginova » et le mot de passe « aginova1234 ». Les capteurs vont par défaut rechercher le réseau sur les canaux 1, 6 ou 11. Ensuite, lorsqu'ils se seront authentifiés auprès du réseau, ils vont envoyer par défaut leurs paquets à l'adresse 192.168.0.10. C'est pourquoi il faut que la Wibox Lite ait l'adresse 192.168.0.10.

Il faut installer le système d'exploitation sur le Raspberry Pi comme expliqué dans le chapitre « Démarrage du Raspberry Pi ». Une fois que l'OS est installé, il faut configurer l'interface réseau. Il est important que le Raspberry Pi ait une adresse IP fixe qui sera 192.168.0.10. Cette adresse est l'adresse par défaut de la Wibox. Pour configurer une adresse IP fixe, il faut éditer le fichier interfaces présent à l'emplacement suivant : /etc/network/. Comme expliqué dans le chapitre « Configuration du réseau sans fil sur le Raspberry Pi », ce fichier contient les paramètres de configuration des différentes interfaces réseau. Pour configurer l'interface Wifi, avec l'adresse IP fixe 192.168.0.10, voici la configuration que j'ai effectuée :

```
allow-hotplug wlan0
auto wlan0
iface wlan0 inet static
    wpa-ssid "aginova"
    wpa-psk "aginova1234"
    address 192.168.0.10
    netmask 255.255.255.0
    network 192.168.0.0
    broadcast 192.168.0.255
    gateway 192.168.0.1
```

Figure 65 Configuration d'une adresse IP fixe

La ligne iface wlan0 inet static, indique que l'adresse IP du Raspberry pi sera une adresse statique. Les autres lignes servent à définir les différents paramètres du réseau Wifi comme l'adresse de sous-réseau, l'adresse de broadcast ainsi que l'adresse du point d'accès.

Maintenant que l'interface réseau du Raspberry Pi est configurée, il faut lui transférer l'exécutable du programme de la Wibox Lite ainsi que le fichier de configuration config.txt qui contient les réglages de l'utilisateur concernant le nettoyage de la base de données.

Comme Linux contient différents répertoires contenant certains types de fichiers, il est très important de placer l'exécutable de la Wibox Lite dans le répertoire /bin. En effet, le répertoire /bin contient tous les exécutables des différentes applications installées sur le système d'exploitation, l'exécutable de la Wibox Lite a donc bien sa place ici. Ensuite, il ne faut pas oublier de compléter les préférences de l'utilisateur dans le fichier config.txt, ces préférences concernent notamment le nettoyage de la base de données. Ce fichier est à placer dans le répertoire /etc. Ce répertoire contient tous les fichiers de configurations des différents programmes fonctionnant sous Linux.

Maintenant que le fichier de configuration est créé et que l'exécutable du programme a été placé dans le répertoire adéquat, il faut encore placer le script de démarrage du programme nommé Wibox\_Lite.sh dans le répertoire /etc/init.d. Comme expliqué dans le chapitre « Lancement du programme au démarrage du Raspberry Pi », ce script sert à lancer le programme de la Wibox\_Lite au démarrage. Pour que le script soit exécuté lors du démarrage du Raspberry Pi, il faut encore exécuter la commande `update-rc.d Wibox_Lite.sh defaults`. Cette commande insère l'exécution du script Wibox\_lite.sh lors de la séquence de démarrage de l'appareil.

Maintenant que le programme est prêt au niveau du Raspberry Pi et que le point d'accès Wifi est correctement configuré, il est encore nécessaire de s'assurer que le Portal fonctionne bien sur la machine. Pour vérifier le bon fonctionnement du Portal, il suffit d'ouvrir un navigateur sur la machine et de taper l'adresse `127.0.0.1/aginova/login.jsp`. Sur cette page, l'utilisateur est invité à entrer le login et le mot de passe pour se connecter au Portal. La valeur par défaut du login est « admin » et le password est « admin1234 ».

Une fois que ces informations ont été entrées, l'utilisateur est connecté au Portal. Il faut maintenant ajouter la gateway, c'est-à-dire la Wibox Lite qui va envoyer les informations des capteurs au Portal Aginova. Pour créer une Gateway, il faut aller sous l'onglet « Administration » et sélectionner « Gateway ».

Lors de la création de la Wibox, nous arrivons sur la page suivante :

Home Sensors Infrastructure Administration

Create/edit a gateway

Make sure that the WiBox or RTLS you are trying to connect to is currently running !

Name\*: Wibox\_Lite Enabled\*: Yes

Host\*: 192.168.0.10 Port\*: 8088

Connect to\*: WiBox / iBox Server Protocol\*: Http

Web Services Username\*: ws Web Services Password\*: .....

(\*) Required

Save

Figure 66 Création d'une Gateway

Lors de la création de la Gateway, il faut entrer différents paramètres comme le nom de la Gateway, si celle-ci est activée ou non ainsi que l'adresse IP de cette dernière, qui sera bien sûr celle du Raspberry Pi, et le port sur lequel le Portal va envoyer les requêtes http qui par défaut est le port 8088. Ensuite, il faut spécifier le type de Gateway à utiliser dans le cas c'est Wibox/Wibox Server et le protocole utilisé est http. La dernière chose à configurer est la communication avec la base de données. En effet, la base de données doit être installée en même temps que le Portal sur la même machine que le Portal et pour pouvoir accéder à cette base de données, il faut entrer le username de la base de données qui par défaut est « ws » et le password qui par défaut est « ws79013 ».

Maintenant que tout ceci est fait, il est nécessaire de s'assurer que l'ordinateur sous lequel fonctionne le Portal et le Raspberry Pi puisse bien communiquer ensemble, pour cela, il est nécessaire de désactiver le pare-feu de l'ordinateur et d'effectuer un ping sur l'adresse IP du Raspberry Pi afin de voir si l'on peut recevoir des données de sa part.

Une fois que tout est en place, le capteur qui va être enclenché va s'authentifier auprès du réseau et va ensuite entamer une communication avec la Wibox Lite. La Wibox Lite va ainsi acquitter les paquets de données auprès du capteur, enregistrer ces données dans une base de données SQLite, qui va être créée dans le dossier /home du raspberry pi, et aussi envoyer les données du capteur au Portal au travers du réseau Wifi. Si tout ceci se passe bien, le capteur enclenché devrait apparaître comme étant en ligne dans l'onglet « Home » du Portal comme on peut le constater ci-dessous :



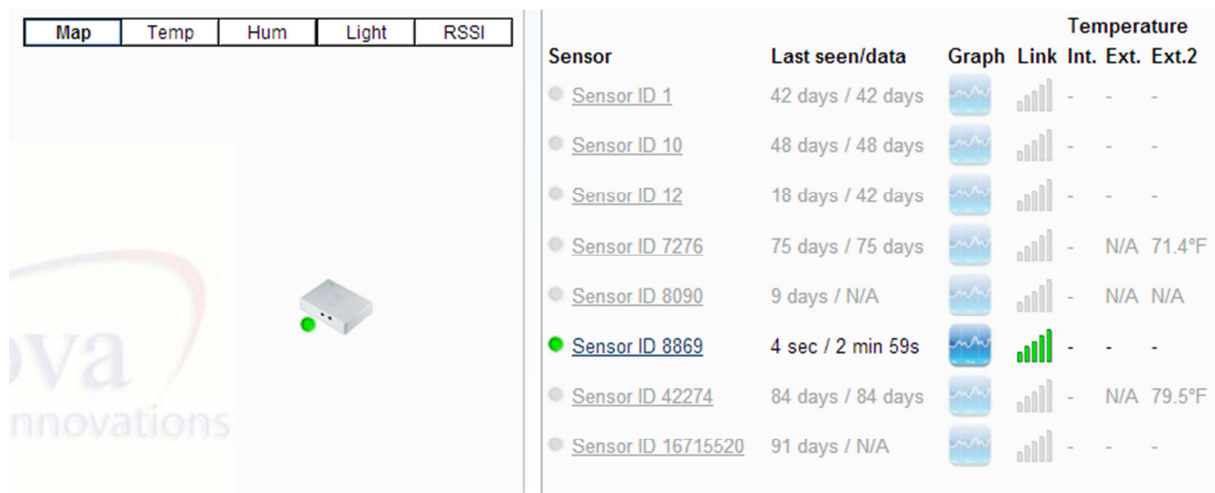


Figure 67 Capteur connecté à la Wibox Lite

#### 4.1 Reconfiguration des capteurs depuis un pc :

Maintenant que la Wibox Lite fonctionne correctement, et que les capteurs apparaissent comme étant en ligne, il faut encore pouvoir reconfigurer différents paramètres du capteur au travers du réseau Wifi. Pour ce faire, j'ai créé le programme `server_tcp.c`. Si le programme est exécuté sans paramètre, il va se connecter automatiquement au port 4098 de l'adresse 192.168.0.10 (adresse par défaut de la Wibox). Il est également possible de spécifier l'adresse de la gateway à laquelle se connecter en passant simplement l'adresse lors du lancement du programme, par exemple, `./server_tcp 192.168.0.5`. Ainsi, si le programme arrive à se connecter à l'adresse indiquée, un menu contenant les différents paramètres à reconfigurer va apparaître dans la ligne de commande :

```
ip address: 192.168.0.10
Connection à 192.168.0.10 sur le port 4098
Menu des commandes:
1: Set sampling period
2: Set sampling sending rate
3: Set config period
4: Set Wibox ip address
5: Set SNMP data port
6: Set pref SSID
7: Set second SSID
8: Set third SSID
9: DHCP On
10: DHCP Off
11: Change sensor ID
12: Quit program
Entrer le numero de votre commande:
```

Figure 68 Menu de reconfiguration des capteurs

Lorsque le menu apparait, il suffit de sélectionner une commande à exécuter en tapant simplement le numéro de la commande. Lorsque le numéro de la commande a été choisi, un nouveau message va s'afficher à l'écran demandant le paramètre de la commande choisi, c'est-à-dire la valeur à attribuer à la commande sélectionnée. Une fois que tous les paramètres de la commande ont été entrés par l'utilisateur, la commande est envoyée au programme de la Wibox Lite qui va se charger de transmettre la requête de reconfiguration au capteur.

La commande 11 « Change sensor ID », permet d'entrer le numéro du capteur que l'on veut reconfigurer. Comme cette version de la Wibox ne communique qu'avec un seul capteur, cette option ne sert à rien pour l'instant. Néanmoins, une champ nommé `id_capteur` est présent dans la structure Commande afin de savoir quel capteur est sélectionné par l'utilisateur. Cette option pourra être utilisée dans une version future de la Wibox lite qui communiquera avec plusieurs capteurs en même temps.

Pour quitter le programme de reconfiguration du capteur, il est nécessaire d'entrer la commande 12 afin que le programme puisse fermer la communication socket avant de se terminer. Ceci permet d'éviter tout bug lors de la fin de la communication avec la Wibox lite.

## 5 Conclusion :

Maintenant que je suis à la fin de mon projet je constate que j'ai rempli les points suivants du cahier des charges : la Wibox lite fonctionne correctement sur le Raspberry pi, elle peut lire et interpréter les différents paquets en provenance des capteurs, elle intègre une base de données lite et peut communiquer avec le Portal grâce au protocole XML/http. Un programme de reconfiguration des capteurs a aussi été créé, celui-ci permet la reconfiguration de différents paramètres propres aux capteurs.

Je n'ai malheureusement pas pu rendre le système compatible avec une carte USB 3G et je n'ai pas pu afficher les données basic à propos de l'état des capteurs, comme des graphs de la batterie, par manque de temps. La communication avec le Portal m'a également pris beaucoup de temps car intégrer un serveur http dans mon programme en C n'était pas une chose facile à faire.

Les améliorations possible de la Wibox lite sont bien sûr de pouvoir rendre le système compatible avec une carte USB 3G pour le remote monitoring mais aussi de pouvoir gérer la communication avec plusieurs capteurs en même temps. Je pense également qu'il serait bien de pouvoir utiliser le Raspberry pi comme point d'accès mais cela nécessiterait sûrement de devoir changer sa distribution.

En conclusion, je tiens à dire que j'ai trouvé ce projet extrêmement intéressant, je n'avais jamais fait de projet aussi conséquent et j'ai vraiment apprécié la partie réalisation. J'ai eu quelques difficultés au début car tous les protocoles étaient nouveaux pour moi et j'ai dû lire beaucoup de documentation avant de pouvoir commencer la programmation.

Je ne connaissais pas du tout le protocole SNMP et j'ai appris à très bien le connaître, car je l'ai utilisé tout au long du projet pour la communication bas niveau avec les capteurs. J'ai également appris à utiliser CVS qui est un logiciel extrêmement utile lors de la réalisation d'un projet. Et j'ai fait beaucoup de programmation embarquée ce qui m'a beaucoup plu, car les compétences requises pour ce projet correspondent tout à fait à ma formation.

Je tiens à remercier Stephan Robert, Karl Baumgartner et Thierry Jayet qui m'ont encadré et ont répondu à mes questions tout au long de mon travail de bachelor. J'ai aussi beaucoup apprécié d'avoir pu finir ce travail à San José State University, cette expérience a été extrêmement intéressante et enrichissante pour moi, même si il était parfois difficile d'envoyer du matériel ou de communiquer à cause du décalage horaire.

Date : 1 août 2013

Signature : Jonathan Despraz



## 6 Bibliographie

- [1] INSTALLATION TOOLCHAIN. Chicoree. [en ligne]. [http://www.chicoree.fr/w/Compilation\\_crois%C3%A9\\_facile\\_pour\\_Raspberry\\_Pi#Installer\\_crosstool-ng](http://www.chicoree.fr/w/Compilation_crois%C3%A9_facile_pour_Raspberry_Pi#Installer_crosstool-ng) [page consultée le 15/07/2013].
- [2] KATE. Wikipedia. [en ligne]. [http://fr.wikipedia.org/wiki/Kate\\_\(logiciel\)](http://fr.wikipedia.org/wiki/Kate_(logiciel)) [page consultée le 18/06/2013].
- [3] LANCER PROGRAMME AUTOMATIQUEMENT. Pihomeserver. [en ligne]. <http://www.pihomeserver.fr/2013/05/27/raspberry-pi-home-server-lancer-un-programme-automatiquement-au-demarrage/> [page consultée le 25/07/2013].
- [4] MIB EDITOR. Webnms. [en ligne]. [http://www.webnms.com/cagent/help/snmp/c\\_snmp\\_mibeditor.html#createnewmib](http://www.webnms.com/cagent/help/snmp/c_snmp_mibeditor.html#createnewmib) [page consultée le 27/03/2013].
- [5] NET-SNMP DOWNLOAD. Net-SNMP. [en ligne]. <http://net-snmp.sourceforge.net/download.html> [page consultée le 01/04/2013].
- [6] NET-SNMP TUTORIAL. Net-SNMP. [en ligne]. <http://net-snmp.sourceforge.net/tutorial/tutorial-5/toolkit/> [page consultée le 01/04/2013].
- [7] PRESENTATION DU PROTOCOLE SNMP. Developpez.com. [en ligne]. <http://ram-0000.developpez.com/tutoriels/reseau/SNMP/> [page consultée le 22/03/2013].
- [8] RASPBERRY PI. Raspberry pi. [en ligne]. <http://www.raspberrypi.org/> [page consultée le 17/05/2013].
- [9] RASPBERRY PI. Wikipedia. [en ligne]. [http://en.wikipedia.org/wiki/Raspberry\\_Pi](http://en.wikipedia.org/wiki/Raspberry_Pi) [page consultée le 17/05/2013].
- [10] RAWCAP. Netresec. [en ligne]. <http://www.netresec.com/?page=RawCap> [page consultée le 07/07/2013].
- [11] SERVER HTTP. Abhijeetr. [en ligne]. <http://blog.abhijeetr.com/2010/04/very-simple-http-server-written-in-c.html> [page consultée le 01/07/2013].
- [12] SIMPLE NETWORK MANAGEMENT PROTOCOL. Wikipedia. [en ligne]. <http://en.wikipedia.org/wiki/SNMP> [page consultée le 22/03/2013].
- [13] SNMP COMMAND EXAMPLES. Docs Oracle. [en ligne]. [http://docs.oracle.com/cd/E19469-01/820-6413-13/SNMP\\_commands\\_reference\\_appendix.html#50446362\\_61509](http://docs.oracle.com/cd/E19469-01/820-6413-13/SNMP_commands_reference_appendix.html#50446362_61509) [page consultée le 05/04/2013].
- [14] SNMP OVERVIEW. Webnms. [en ligne]. [http://www.webnms.com/cagent/help/technology\\_used/c\\_snmp\\_overview.html](http://www.webnms.com/cagent/help/technology_used/c_snmp_overview.html) [page consultée le 22/03/2013].

- [15] SQLITE WITH C. Manish's Tech Blog. [en ligne]. <http://milky.manishsinha.net/2009/03/30/sqlite-with-c/> [page consultée le 07/06/2013].
- [16] SQLITE. Code FAQ. [en ligne]. <http://fr.softuses.com/123174> [page consultée le 04/07/2013].
- [17] SQLITE. SQLite. [en ligne]. <http://www.sqlite.org/> [page consultée le 31/05/2013].
- [18] SQLITE. Wikipedia. [en ligne]. <http://fr.wikipedia.org/wiki/SQLite> [page consultée le 31/05/2013].
- [19] STATIC IP ADDRESS ON RASPBERRY PI. Elinux. [en ligne]. [http://elinux.org/FR:Configuring\\_a\\_Static\\_IP\\_address\\_on\\_your\\_Raspberry\\_Pi](http://elinux.org/FR:Configuring_a_Static_IP_address_on_your_Raspberry_Pi) [page consultée le 29/07/2013].
- [20] USING AND LOADING MIBS. Net-SNMP. [en ligne]. [http://www.net-snmp.org/wiki/index.php/TUT:Using\\_and\\_loading\\_MIBS](http://www.net-snmp.org/wiki/index.php/TUT:Using_and_loading_MIBS) [page consultée le 09/04/2013].
- [21] VALGRIND. Unixgarden. [en ligne]. <http://www.unixgarden.com/index.php/gnu-linux-magazine/corriger-votre-utilisation-memoire-avec-valgrind> [page consultée le 17/07/2013].

## 7 Annexes

### 7.1 Journal de travail

**Date séance : 12 mars 2013**

Place de travail : Bureau Aginova

09h15 :

Explication du principe de fonctionnement de la Wibox ainsi que du Portal. Définition du projet à effectuer dans les grandes lignes.

10h00 :

Je reçois les différentes marche à suivre ainsi que les procédures nécessaire pour installer les logiciels Wibox et Portal.

12h00 :

Fin de l'installation des différents logiciels, je peux recevoir les différentes données mesurées par les capteurs grâce au Portal.

13h30 :

Je commence à découvrir la distribution d'OpenWrt présente sur la carte netcore NW708 que l'on m'a confiée pour le projet.

14h00 :

Communication avec la carte et recherche sur internet de différentes informations nécessaires à la compilation de mon premier programme sur cette carte embarquée.

15h00 :

Téléchargement du SDK nécessaire afin de cross compiler un programme pour la plateforme cible. Il m'a fallu pas mal de temps afin de trouver quelle version du SDK utilisé et pour trouver comment il est possible de créer un exécutable pour la machine cible.

15h30 :

Réalisation d'un tutoriel trouvé sur le site suivant : [http://www.gargoyle-router.com/wiki/doku.php?id=openwrt\\_coding](http://www.gargoyle-router.com/wiki/doku.php?id=openwrt_coding). Puis réalisation d'un programme de test afin de pouvoir mettre en place l'environnement. Il est encore nécessaire que j'installe plusieurs composants sur ma machine virtuelle linux afin de pouvoir cross-compiler le programme réalisé.

16h30 :

Fin de séance.

**Date séance : 13 mars 2013**

Place de travail : Domicile

09h00 :

Installations des différents paquets permettant la compilation de fichier pour la plateforme embarquée utilisée.

09h30 :

Création d'un package contenant un simple programme de test ainsi que deux Makefiles permettant la compilation de ce package.

11h30 :

La compilation s'est correctement passée, je transfère et installe le package que j'ai créé sur la carte afin de tester son fonctionnement.

12h00 :

Mon programme s'exécute correctement sur la carte.

Fin de séance.

### **Date séance : 15 mars 2013**

Place de travail : Bureau Aginova

09h15 :

Reprise du projet et recherche concernant la plateforme embarquée. Je remarque que le wifi ne fonctionne pas sur la carte malgré un SoC Ralink RT3050F qui contient tout le hardware nécessaire pour la communication Wifi. C'est très bizarre car les drivers semblent être installés.

10h30 :

Je remarque le message suivant lors de l'initialisation de la carte :

phy0 -> rt2800\_init\_eeprom: Error - Invalid RF chipset detected.

phy0 -> rt2x00lib\_probe\_dev: Error - Failed to allocate device.

Ce qui laisse penser qu'il y a clairement un problème de driver c'est pourquoi j'essaie de réinstaller les drivers et de les remettre à jour.

13h30 :

Les drivers ne fonctionnent toujours pas et le problème persiste après plusieurs recherches, je suis arrivé sur des sites qui contiennent des mises à jour software pour la carte NetCore NW708.

14h30 :

J'effectue les différentes mises à jour mais malheureusement après le reboot la carte ne semble plus fonctionner.

15h00 :

Après quelques discussions avec Karl Baumgartner, celui-ci me conseille de me concentrer sur l'analyse des paquets avec Wireshark et la compréhension du protocole de communication entre les paquets et l'AP.

16h30 :



Fin de séance.

**Date séance : 18 mars 2013**

Place de travail : Domicile

19h00 :

Début de séance je démarre les capteurs et connecte le point d'accès à mon PC. Il est nécessaire de donner l'adresse IP 172.19.0.10 à l'ordinateur ce qui est l'adresse de destination des paquets envoyés par les capteurs.

19h30 :

J'arrive à lire les paquets wireshark sur Windows 7 mais je n'arrive pas à les lire à l'aide de ma machine virtuelle. Il est nécessaire de changer la configuration du réseau de la machine virtuelle comme je l'ai expliqué dans le rapport afin de pouvoir recevoir les paquets UDP sur la machine virtuelle Linux. Il est également nécessaire de désactiver le firewall afin de pouvoir recevoir ces paquets.

20h30 :

Je commence à analyser la structure des paquets UDP arrivant sur le port 162 comme définit dans la documentation. Ces paquets utilisent le protocole SMNP grâce à wireshark je peux voir les données qu'ils contiennent.

21h00 :

Je commence à écrire un programme en C utilisant les sockets UDP afin de pouvoir recevoir et traiter les paquets et récupérer les informations contenues dans ceux-ci.

22h00 :

Fin de séance.

**Date séance : 19 mars 2013**

Place de travail : Domicile

19h00 :

Je remets en place le point d'accès et démarre les capteurs je dois également refaire les manipulations expliquées ci-dessus afin de pouvoir à nouveau analyser les paquets wireshark sur la machine virtuelle.

19h30 :

Je continue à écrire le programme en C destiné à analyser les paquets UDP.

20h30 :

J'arrive à afficher le voltage des différents capteurs ainsi que l'information clock qui est contenue dans ceux-ci.

**Date séance : 22 mars 2013**

Place de travail : Bureau Aginova

09h15 :

Je reprends le travail et demande à Karl comment faire pour que les capteurs envoient directement des paquets de données à l'AP sans créer de connexion pour autant. Karl règle les capteurs et m'explique comment faire pour déchiffrer les paquets de données envoyés à destination de l'AP.

12h00 :

J'arrive à déchiffrer les paquets de données provenant des capteurs et ainsi à savoir leur température différentes informations comme leur IDs.

13h15 :

Karl m'explique que les capteurs et l'AP utilisent le protocole SNMP (Simple Network Management Protocol). Il me conseille de lire la documentation sur ce protocole afin de pouvoir communiquer depuis mon programme avec les capteurs. Je commence donc à lire la documentation.

16h30 :

Fin de séance.

**Date séance : 27 mars 2013**

Place de travail : HEIG-VD

13h00 :

Je continue à lire la documentation sur le protocole SNMP et essaye de comprendre comment fonctionnent les fichiers MIB définissant les différents objets pris en compte par le protocole.

16h00 :

Analyse d'exemple de programme en C utilisant le protocole SNMP afin de communiquer avec différents objets à travers le réseau sans fil. Il existe plusieurs fonctions dont j'essaie de comprendre le fonctionnement.

17h00 :

Fin de séance

**Date séance : 1 avril 2013**

Place de travail : Domicile

09h00 :

Je continue à lire de la documentation sur le protocole SNMP et les fichiers MIB. Je commence à écrire un fichier MIB contenant les différents object identifier pour les capteurs Aginova.

12h00 :

J'arrive à envoyer des requêtes SNMP grâce au programme en C que j'ai développé sur Linux.

13h00 :

Analyse Wireshark des paquets envoyé par la Wibox au capteur afin d'analyser leur structure et comprendre les différentes informations dont ils sont constitués.

16h00 :

Fin de séance.

**Date séance : 5 avril 2013**

Place de travail : Bureau Aginova

09h00 :

Karl me parle des différents paquets SNMP qui sont échangés entre la Wibox et les capteurs il m'explique comment se passe l'acquittement des paquets qui est fait au travers de variables dans les requêtes SNMP.

10h00 :

Je commence donc à lire la documentation afin de pouvoir envoyer des requêtes SNMP contenant des variables.

13h00 :

J'arrive à envoyé des requêtes contenant des variables. Karl m'explique alors que la variable doit contenir le numéro du paquet qui est acquitté par la Wibox. Je commence donc à écrire un programme nommé `snmp_communication.c` qui se charge d'acquitter les paquets envoyés par le capteur.

17h00 :

Fin de séance.

**Date séance : 9 avril 2013**

Place de travail : Domicile

13h00 :

Je continue à écrire le programme `snmp_communication.c` qui doit permettre d'acquitter les paquets envoyés par le capteur il est nécessaire de travailler avec les sockets afin de traités les paquets de données et d'en extraire le numéro pour pouvoir l'acquitter au travers du protocole SNMP.

17h00 :

Fin de séance.

**Date séance : 12 avril 2013**

Place de travail : Bureau Aginova

09h00 :

J'explique l'avancement du programme que j'ai réalisé au cours de la semaine précédente à Karl qui m'explique qu'il faut que je découpe le programme en plusieurs threads. Il m'explique aussi qu'il faudrait pouvoir donner des commandes au programme afin de pouvoir reconfigurer le capteur.

12h00 :

Le programme est découpé en différents threads qui s'occupent de différentes tâches comme l'écoute des données sur un port et l'envoi des requêtes SNMP sur un autre port.

13h00 :

Je constate qu'il n'y a pas d'utilitaire pour recevoir des traps SNMP dans un programme C or il est important que je puisse récupérer des traps SNMP qui sont en faite des messages nommés configuration update et qui signifient que le capteur attend un acquittement des paquets envoyés.

15h00 :

Je constate qu'un programme nommé snmptrapd est fourni avec la suite net-snmp et ce logiciel permet d'exécuter des scripts à la réception de traps snmp. Je commence donc à configurer ce logiciel afin de pouvoir exécuter différentes opérations à la réception de traps SNMP.

17h00 :

Fin de séance.

**Date séance : 15 avril 2013**

Place de travail : Domicile

18h00 :

Rédaction du rapport.

12h00 :

Fin de séance

**Date séance : 16 avril 2013**

Place de travail : Domicile

13h00 :

Je réalise un programme annexe qui est chargé d'envoyer des commandes au programme au programme de communication SNMP. Ce programme sera utilisé pour envoyer différentes commandes aux capteurs notamment pour modifier leur configuration etc...

16h00 :

Je modifie le programme de communication SNMP pour qu'il soit capable de recevoir les différentes informations du programme annexe servant à transmettre différentes commandes aux capteurs. Pour la communication entre ces deux programmes, j'utilise la communication socket à travers le protocole TCP ce qui permet de pouvoir utiliser ces deux programmes de communications sur deux ordinateurs différents se trouvant dans le même sous-réseau.

18h00 :

Fin de séance.

**Date séance : 17 avril 2013**

Place de travail : Bureau Aginova

09h00 :

Je commence mes recherches sur le logiciel snmptrapd qui doit permettre de capturer des trappes SNMP et de pouvoir exécuter des commandes Linux à la réception de celles-ci.

11h00 :

Je configure le fichier snmptrapd.conf qui permet de paramétrer le logiciel snmptrapd.

13h00 :

J'écris un script bash qui va permettre de lancer un programme lors de la réception de certaines trappes SNMP. Ce programme fonctionne communiqué à l'aide de la programmation socket.

15h00 :

Karl m'explique les différentes fonctions que mon programme devra intégrer notamment les fonctions de reconfiguration des capteurs.

16h00 :

La configuration du logiciel snmptrapd est finie et je peux maintenant lancer mon script à la réception des différentes trappes SNMP.

17h00 :

J'ajoute un thread nommé trap\_listener dans le programme de communication avec les capteurs afin d'attendre qu'une trappe ait été réceptionnée avant d'effectuer l'acquittement des paquets par exemple.

**Date séance : 20 avril 2013**

Place de travail : Domicile

10h00 :

Rédaction du rapport et avance sur le protocole de communication qui va me permettre d'envoyer des commandes d'un programme à l'autre.

15h00 :

Fin de séance.

**Date séance : 24 avril 2013**

Place de travail : Domicile

12h00 :

Je continue la programmation et je fais des captures wireshark afin d'étudier les valeurs qui sont contenues variables SNMP lors des changements de sampling period et heartbeat period.

17h00 :

J'arrive à transmettre d'un programme à l'autre un changement de sampling period ou heartbeat period grâce à une structure contenant un oid ainsi qu'une valeur de variable. Il me faut encore transmettre ces informations aux capteurs à l'aide d'une requête SNMP.

Fin de séance.

**Date séance : 26 avril 2013**

Place de travail : Bureau Aginova

9h00 :

Je reprends le développement du programme qui va me permettre de reconfigurer les différentes périodes des capteurs. Karl me dit que les informations sur les valeurs contenues dans les requêtes SNMP se trouvent dans un fichier nommé « Sensor communication-wifi.docx ».

14h00 :

J'arrive maintenant à envoyer des requêtes afin de reconfigurer les différentes périodes du capteur. Par contre le capteur ne s'affiche plus dans la Wibox.

17h00 :

Karl reprogramme le capteur car l'ID du capteur a été changé lors de la reconfiguration du capteur par mon programme c'est pour cela que le capteur n'apparaissait pas sur la Wibox. Karl va me communiquer les valeurs qu'il faut mettre par défaut dans les champs de reconfiguration afin de ne pas modifier l'ID des capteurs.

17h30 :

Fin de séance.

**Date séance : 29 avril 2013**

Place de travail : Domicile

13h00 :

Rédaction du chapitre du rapport concernant le principe de fonctionnement de la Wibox

17h00 :

Fin de séance

**Date séance : 30 avril 2013**

Place de travail : Domicile

13h00 :

Rédaction du chapitre du rapport concernant les différents types de données échangées entre les capteurs et la Wibox.

17h00 :

Fin de séance

**Date séance : 3 mai 2013**

Place de travail : Bureau Aginova

9h15 :

Je continue la rédaction du programme chargé d'envoyer les différentes commandes au programme de communication avec les capteurs.

12h00 :

Il est nécessaire de reprogrammer le capteur car l'envoi de différentes données de reconfiguration ont modifié son ID et la Wibox ne l'identifie plus.

13h00 :

Je continue la rédaction du programme et implémente les fonctions permettant de changer la sampling period, heartbeat period ainsi que sampling rate.

17h00 :

Fin de séance.

**Date séance : 6 mai 2013**

Place de travail : Domicile

13h00 :

Rédaction du programme servant à envoyer les commandes. Je choisis de chaîner les commandes afin de pouvoir en envoyer plusieurs à la fois.

17h00 :

Fin de séance

**Date séance : 9 mai 2013**

Place de travail : Domicile

9h00 :

Fin de la rédaction du programme servant à envoyer des commandes au programme de communication avec le capteur.

13h00 :



Je commence à modifier le programme de communication avec les capteurs afin qu'il puisse recevoir les commandes envoyées par le programme d'envoi des commandes. Il est nécessaire de reconstruire une liste chaînée avec les commandes reçues.

17h00 :

Fin de séance

**Date séance : 10 mai 2013**

Place de travail : Domicile

9h00 :

Implémentations des différentes fonctions de la Wibox présentes dans l'onglet « Wibox Parameters » et « TCP/IP Parameters ».

17h00 :

Fin de séance.

**Date séance : 12 mai 2013**

Place de travail : Domicile

9h00 :

Je continue à écrire le programme de communication du côté du serveur et du côté du programme de communication avec les capteurs.

12h00 :

Il y a toujours quelques problèmes au niveau de la reconstruction de la chaîne de commandes dans le programme qui réceptionne les commandes transmises par TCP

17h00 :

Fin de séance.

**Date séance : 14 mai 2013**

Place de travail : Domicile

12h00 :

Je continue à travailler sur le programme qui envoie et réceptionne les différentes commandes pour la reconfiguration du capteur par le réseau Wifi. Implémentation de la reconfiguration du protocole DHCP au niveau des capteurs.

16h00 :

La liste chaînée est reconstituée dans le programme de communication avec les capteurs mais il reste à implémenter la destruction de cette chaîne (free) après l'envoi des différentes commandes qu'elle contient.

18h00 :

Fin de séance.

**Date séance : 17 mai 2013**

Place de travail : Bureau Aginova

09h00 :

Je termine les différentes fonctionnalités à implémenter au niveau des capteurs, Karl me fait remarquer que j'ai oublié d'envoyer la commande de reset lors de l'envoi de certaines commandes comme lors de la reconfiguration de l'adresse IP d'un capteur. Je travaille pour corriger ce problème.

13h00 :

Réception du Raspberry pi sur lequel je vais devoir exécuter mon programme et lecture de la documentation et du guide de démarrage de celui-ci.

15h00 :

Je copie l'OS du Raspberry pi sur une carte SD afin de pouvoir le démarrer. Comme il n'y a pas d'écran HDMI ni VGA je suis obligé d'accéder au Raspberry pi via Ethernet à l'aide du protocole SSH.

17h30 :

Fin de séance.

**Date séance : 21 mai 2013**

Place de travail : Domicile

15h00 :

Commande des différentes pièces manquantes pour le Raspberry pi, il a été nécessaire de commander un dongle Wifi usb ainsi qu'un adaptateur pour passer d'un connecteur HDMI à un connecteur VGA.

16h00 :

Fin de séance.

**Date séance : 26 mai 2013**

Place de travail : Domicile

09h00 :

Je branche le Raspberry pi à la télévision et je configure les différentes choses nécessaires à son bon fonctionnement, il faut notamment configurer le clavier la langue ainsi que différents paramètres réseau

12h00 :

Je commence à installer les différentes bibliothèques nécessaires au bon fonctionnement du programme de communication avec les capteurs sur le Raspberry pi.

16h00 :

Fin de séance.

**Date séance : 27 mai 2013**

Place de travail : Domicile

15h00 :

J'installe encore les différents utilitaires Net-SNMP sur le Raspberry pi et transfère mon programme de communication sur ce dernier.

17h00 :

Il est encore nécessaire de configurer le script de configuration SNMP sur le Raspberry pi afin que les paquets SNMP soient correctement traités.

20h00 :

Fin de séance.

**Date séance : 29 mai 2013**

Place de travail : Domicile

17h00 :

Le programme de communication fonctionne maintenant correctement sur le Raspberry pi si celui-ci est directement câblé au routeur. Je configure maintenant le programme d'envoi des commandes afin que celui-ci envoie directement les commandes au Raspberry pi et non pas au localhost.

20h00 :

Fin de séance le programme d'envoi des commandes communique correctement avec le Raspberry pi.

**Date séance : 31 mai 2013**

Place de travail : Bureau Aginova

09h15 :

Réception de l'adaptateur HDMI to DVI et du dongle Wifi pour le Raspberry pi. La première chose à faire est donc de brancher le Raspberry pi sur un écran et de connecter ce dernier au réseau Wifi.

11h00 :

Le Raspberry pi se connecte sans problème au réseau Wifi et fonctionne selon mes attentes. Je décide maintenant d'implémenter la base de données SQLite dans le programme de communication.

16h30 :

Installation de l'outil CVS qui est un outil de gestion des versions. Karl nous explique comment utiliser ce logiciel et nous mettons nos projets sur le serveur.

17h30 :

Fin de séance.

**Date séance : 1 juin 2013**

Place de travail : Domicile

10h00 :

Rédaction du rapport de travail de Bachelor.

15h00 :

Fin de séance.

**Date séance : 3 juin 2013**

Place de travail : HEIG-VD

12h00 :

Rédaction du rapport de travail de Bachelor.

16h00 :

Fin de séance.

**Date séance : 5 juin 2013**

Place de travail : Domicile

18h00 :

Maintenant que la communication SNMP avec le capteur fonctionne bien, je décide de désactiver le service de la Wibox et d'implémenter la procédure de connexion avec le capteur.

21h30 :

Je remarque que pour que l'envoi de la première requête puisse se faire avec le capteur, il faut remplacer la valeur du champ community par « GSN\_GET » au lieu de « GSN\_SET ».

22h30 :

La partie d'initialisation de la communication avec les capteurs fonctionne parfaitement. Fin de séance.

**Date séance : 7 juin 2013**

Place de travail : Bureau Aginova

09h00 :

Je discute avec Karl de la structure que doit avoir la base de données et nous faisons un croquis du travail à faire.

10h00 :

Karl m'explique aussi qu'il faut extraire les données de paquet de heartbeat afin de stocker les données qu'ils contiennent dans la base de données.

14h00 :

J'ai créé un thread nommé `database_cleaner` qui va se charger d'analyser les données stockées dans la base de données et d'éliminer les plus anciennes.

17h30 :

J'ai commencé à changer le système de réception des trappes SNMP afin de pouvoir récupérer les informations contenues dans les messages « Heartbeat » encapsulés dans des trappes SNMP.

Fin de séance.

### **Date séance : 8 juin 2013**

Place de travail : Domicile

10h00 :

Je commence à implémenter le parseur de messages « heartbeat ». J'ai utilisé des fonctions de type string afin de sélectionner les données que je vais devoir extraire.

11h00 :

Il faut créer une structure contenant toutes les informations sur les heartbeat qui va être complétée avec les informations extraites des paquets reçus.

14h00 :

J'arrive à parser les données et à compléter les différents champs de la structure « Heartbeat ». Fin de séance.

### **Date séance : 11 juin 2013**

Place de travail : Domicile

12h00 :

Je commence à lire le document « Wibox API Developer Guide » afin de comprendre les échanges de paquets entre la Wibox et le Portal. Ces échanges sont assez complexes, c'est pourquoi je prends du temps afin de bien comprendre les subtilités du protocole de communication.

16h00 :

Fin de séance.

### **Date séance : 14 juin 2013**

Place de travail : Bureau Aginova

10h00 :

Je continue à lire le document « Wibox API Developer Guide ».

13h00 :

Je commence à faire des captures de paquets envoyés sur le localhost afin de pouvoir capturer les paquets échangés entre la Wibox et le Portal.

15h00 :

J'étudie les paquets et Karl m'explique la procédure de connexion et les différents paquets envoyés depuis la Wibox au Portal. Il m'explique aussi comment pouvoir communiquer avec plusieurs capteurs qui seront disposés derrière un NAT. J'étudie ensuite comment modifier le programme en conséquence.

18h00 :

Fin de séance.

**Date séance : 17 juin 2013**

Place de travail : Domicile

12h00 :

Je continue à lire la documentation sur la partie de communication http/XML avec le serveur. J'analyse également des captures Wireshark que je fais sur le localhost afin de voir comment les connexions sont faites entre la Wibox et le serveur.

17h00 :

Je peaufine le programme en commentant certaines parties qui ont besoin d'être commentées et continue d'écrire le rapport.

19h00 :

Fin de séance.

**Date séance : 19 juin 2013**

Place de travail : Domicile

16h00 :

Je mets en forme le rapport et effectue les dernières modifications j'inclus également les annexes au rapport

19h00 :

Fin de séance.

**Date séance : 25 juin 2013**

Place de travail : King library

9h00 :

Je continue à lire le document « WiBox API Developer Guide 1.17 rev 24 » afin de comprendre le protocole de communication entre la Wibox et le Portal.

16h00 :

Fin de séance.

**Date séance : 26 juin 2013**

Place de travail : King library

9h00 :

Lecture du document « WiBox API Developer Guide 1.17 rev 24 » et je recherche le code d'un serveur http sur internet afin de pouvoir communiquer entre la Wibox et le Portal en voyant des requêtes http.

14h00 :

J'ai téléchargé le serveur http à l'adresse suivante : <http://blog.abhijeetr.com/2010/04/very-simple-http-server-written-in-c.html>. Je commence à analyser ce code afin de comprendre comment le serveur http est implémenté.

16h30 :

Fin de séance.

**Date séance : 27 juin 2013**

Place de travail : King library

9h00 :

Je continue à analyser l'implémentation du serveur http que j'ai téléchargé et regarde aussi les paquets échangés sur le localhost lors de la communication entre le serveur et la Wibox.

13h00 :

Je commence à modifier le code du serveur http afin qu'il puisse envoyer le flux de données en provenance de la Wibox au Portal. Je regarde aussi comment faire la procédure de connexion auprès du Portal (envoi du login et password).

16h00 :

Fin de séance.

**Date séance : 28 juin 2013**

Place de travail : King library

9h00 :

Je modifie toujours le serveur http pour pouvoir envoyer des paquets de Linkup au Portal afin que celui-ci détecte ma Wibox Lite comme étant connectée.



14h00:

La Wibox envoie des paquets de Linkup au serveur et le Portal affiche la Wibox Lite comme connectée.

16h30 :

Fin de séance.

**Date séance : 1 juillet 2013**

Place de travail : King library

9h00 :

Maintenant que la Wibox Lite s'affiche comme connectée et que je n'ai pas encore reçu mon Access Point qui doit venir de Suisse, je décide de compléter le rapport et d'améliorer quelques algorithmes dans mon code afin de l'optimiser.

13h00 :

J'ajoute une option dans le menu de reconfiguration des capteurs. Cette option sert à choisir l'ID du capteur à reconfigurer. Cette option va servir à sélectionner le capteur à reconfigurer lorsque plusieurs capteurs seront connectés à la Wibox.

16h30 :

Fin de séance.

**Date séance : 2 juillet 2013**

Place de travail : King library

8h30 :

Aujourd'hui, je décide d'implémenter le nettoyage de la base de données à partir des données présente dans un fichier compléter par l'utilisateur. Ce fichier comporte différents paramètres comme la période à laquelle le programme va parcourir la base de données ou é partir de combien de temps les données sont considérées comme étant obsolètes.

14h00 :

Je commence à installer les utilitaires nécessaires afin de faire de la compilation croisée pour mon Raspberry Pi. J'écris au fur et à mesure ces différentes étapes dans le rapport.

17h00 :

Fin de séance.

**Date séance : 3 juillet 2013**

Place de travail : King library

8h30 :

Je continue a installé les utilitaires nécessaire à la création de la toolchain pour le Raspberry pi. Je rencontre quelques problèmes au moment de compiler la toolchain notamment à cause de l'utilitaire ClooG que je vais devoir télécharger manuellement.

13h00 :

J'essaye de configurer un Access Point afin que le capteur puisse se connecter dessus et puisse commencer à envoyer ses données, malheureusement, le capteur ne semble pas pouvoir se connecter à l'AP pour une raison inconnue, je passe la fin de l'après-midi à essayer de résoudre ce problème.

17h00 :

Fin de séance.

**Date séance : 5 juillet 2013**

Place de travail : King library

8h30 :

Aujourd'hui je décide d'avancer dans la rédaction du rapport et je décide aussi de peaufiner le code, c'est-à-dire d'ajouter des en-têtes aux threads et fonctions et de mieux expliquer les différentes étapes de la réalisation notamment l'installation de la toolchain pour le Raspberry Pi.

17h00 :

Fin de séance.

**Date séance : 6 juillet 2013**

Place de travail : King library

10h00 :

Comme je n'ai qu'un programme écrit en C et assez minimaliste pour parcourir ma base de données SQLite, je décide d'installer un utilitaire nommé « SQLite Database browser » qui permet de faire beaucoup d'opérations sur les bases de données SQLite. Je décris son fonctionnement dans le rapport et effectue quelques requêtes SQL.

15h00 :

Fin de séance.

**Date séance : 8 juillet 2013**

Place de travail : King library

08h30 :

Je continue à écrire quelques parties du rapport et à améliorer celui-ci. Je détaille également plus le code et améliore quelques algorithmes de celui-ci.

11h00 :

Je commence à travailler sur la procédure de communication entre le Portal et la Wibox Lite. Je constate qu'il faut envoyer certains paquets spécifiques afin de s'authentifier auprès du Portal.

15h00 :

Une alarme feu retentit et nous sommes obligés de quitter la bibliothèque. Je remercie Joël Tinguely d'avoir sauvé mon matériel avant que le personnel nous bloque l'accès à la bibliothèque.

Fin de séance.

**Date séance : 9 juillet 2013**

Place de travail : King library

08h30 :

Je continue à travailler sur la procédure de connexion entre la Wibox et le Portal, j'analyse les paquets que je capture sur wireshark et qui sont échangés entre ces deux entités.

13h00 :

Je décide d'installer les différentes librairies sur le Raspberry pi afin de pouvoir faire tourner le programme de la Wibox sur celui-ci.

17h00 :

La compilation des librairies pour le Raspberry pi prend du temps et il est nécessaire d'apporter quelques modifications au programme de la Wibox afin que celui-ci fonctionne complètement sur le Raspberry pi mais au final le programme fonctionne sans problème.

**Date séance : 10 juillet 2013**

Place de travail : King library

08h30 :

J'ai enfin reçu le matériel dont j'avais besoin c'est-à-dire un AP ainsi que deux capteurs supplémentaires. Grâce à ce matériel, je vais pouvoir travailler sur la communication entre plusieurs capteurs avec la Wibox au travers d'un NAT.

10h00 :

Après avoir configuré l'AP pour que les capteurs puissent se connecter à la Wibox, je mets à jour la base de données relationnelle SQLite en rajoutant quelques champs et en complétant les informations présentes dans le rapport.

13h00 :

Je travaille sur la connexion de plusieurs capteurs à la Wibox ceci prend beaucoup de temps car il est nécessaire de changer quelque peu l'architecture du programme afin de rendre celui-ci un peu plus flexible.

17h00 :

Fin de séance.

**Date séance : 11 juillet 2013**

Place de travail : King library

08h30 :

Je continue à travailler sur la connexion plusieurs capteurs à la Wibox Lite. Il est nécessaire de créer des structures afin d'encapsuler les données propres aux différents capteurs afin de pouvoir répondre correctement à leurs requêtes SNMP.

13h00 :

Je continue de travailler sur l'enregistrement des données envoyées par le capteur dans la base de données SQLite.

17h00 :

Fin de séance

**Date séance : 16 juillet 2013**

Place de travail : King library

12h30 :

Je continue à travailler sur l'implémentation de la base de données je dois enregistrer les données sous forme de texte dans la base de données et non sous forme hexadécimale comme je le faisais jusqu'à maintenant.

13h00 :

Je complète le rapport et me renseigne sur la configuration de l'AP en mode NAT.

17h00 :

Fin de séance.

**Date séance : 17 juillet 2013**

Place de travail : King library

8h30 :

Aujourd'hui je travaille sur l'extraction des données présente dans les paquets de données envoyés par le capteur à la Wibox, il est nécessaire d'en extraire le timestamp, les deux index des données ainsi que le numéro de paquet à acquitter ce qui est déjà fait.

15h00 :

Maintenant que les données sont extraites et mises dans une structure je commence à modifier le serveur http afin que celui-ci puisse envoyer les données reçues du capteur en streaming.

17h00 :

Fin de séance.

**Date séance : 18 juillet 2013**

Place de travail : King library

8h30 :

Je continue à travailler sur la transmission des données des capteurs au Portal et sur le streaming de ces données. Il est nécessaire de passer des structures de données pour chaque capteur ce qui me pose quelques problèmes.

17h00 :

J'ai passé toute la journée à travailler sur l'envoi des données en streaming au Portal et il reste toujours des problèmes sur les structures à envoyer.

**Date séance : 20 juillet 2013**

Place de travail : King library

8h30 :

Aujourd'hui je décide d'avancer dans la rédaction de mon rapport, je décris les logiciels que j'ai utilisés pour mettre au point mon application ainsi que mon environnement de développement et les caractéristiques de ma machine virtuelle. Je relis aussi une partie du rapport et corrige quelques erreurs de rédaction.

17h00 :

Fin de séance.

**Date séance : 21 juillet 2013**

Place de travail : King library

10h30 :

Je commence à travailler sur le contrôle des saisies dans le programme qui envoie les commandes au programme de communication avec les capteurs. Dans ce programme, on peut entrer des adresses IP ainsi que des chiffres pour paramétrer les capteurs et il est important de rendre ce programme robuste afin d'éviter tout bug.

17h00 :

Fin de séance.

**Date séance : 22 juillet 2013**

Place de travail : King library

8h30 :

Comme tout mon programme se trouve seulement dans un fichier, je trouve qu'il est grand temps de créer une librairie contenant le corps de toutes mes fonctions. Je crée donc un fichier

communication\_utils.h et communications\_utils.c qui contiennent le header ainsi que le corps de mes différentes fonctions.

13h00 :

Je continue à travailler sur l'envoi des données au Portal, il est nécessaire que je parse complètement les messages de heartbeat pour pouvoir récupérer toutes les données à envoyer au Portal je passe donc l'après-midi à récupérer ces données afin de les envoyer au Portal.

16h30 :

Fin de séance.

**Date séance : 23 juillet 2013**

Place de travail : King library

8h30 :

Je remarque que les données envoyées au serveur sont toujours les mêmes et je ne comprends pas pourquoi car elles ces données sont dans variable globale et donc accessible par tous les threads. Je passe un grand moment à essayer de résoudre ce problème jusqu'à ce que je me rende compte que j'utilise l'appel système fork() qui va dupliquer les processus et ainsi copier les variables globales. Je décide donc d'enlever l'appel système fork() et remodèle le programme en fonction.

15h00 :

Je continue l'écriture du rapport et détaille les opérations que j'ai effectuées pour corriger ce bug.

17h00 :

Fin de séance.

**Date séance : 24 juillet 2013**

Place de travail : King library

8h30 :

Je décide maintenant de contrôler le fonctionnement du programme de reconfiguration des capteurs et je constate que j'ai oublié d'implémenter la reconfiguration du réseau Wifi sur lequel les capteurs vont se connecter. Je passe donc toutes la journée à implémenter la reconfiguration des trois réseaux Wifi auxquels les capteurs vont se connectés.

17h00 :

Fin de séance.

**Date séance : 25 juillet 2013**

Place de travail : King library

8h30 :

Il est temps de mettre le programme sur le Raspberry Pi et tester le fonctionnement de celui-ci, je décide de créer un script qui va lancer le programme au démarrage sur le Raspberry Pi et je décide aussi d'écrire dans le rapport un mode d'emploi qui explique comment déployer l'application sur le Raspberry Pi afin que celle-ci se lance automatiquement.

15h00 :

Je mets en forme les paragraphes du rapport que j'ai écrit précédemment.

17h00 :

Fin de séance.

**Date séance : 26 juillet 2013**

Place de travail : King library

8h30 :

Je constate qu'il reste quelques bugs dans l'application notamment lorsque je change l'adresse IP du capteur je travaille tout le jour à implémenter la correction de ces bug.

17h00 :

Fin de séance.

**Date séance : 27 juillet 2013**

Place de travail : King library

11h00 :

Je continue à écrire le mode d'emploi expliquant comment déployer mon application sur une plateforme embarquée et je configure l'interface réseau du Wifi du Raspberry pi afin que celui-ci ait une adresse fixe 192.168.0.10.

17h00 :

Fin de séance.

**Date séance : 29 juillet 2013**

Place de travail : King library

08h30 :

Je continue à écrire le mode d'emploi expliquant comment déployer mon application sur une plateforme embarquée et je configure l'interface réseau du Wifi du Raspberry pi afin que celui-ci ait une adresse fixe 192.168.0.10.

17h00 :

Fin de séance.

**Date séance : 30 juillet 2013**

Place de travail : King library

08h30 :

Je constate que lorsque je change le port d'envoi des données du capteur, le socket continue d'écouter sur le port 163. Je décide donc de changer le port d'écoute des données de la Wibox lite lorsque le port d'envoi des données est reconfiguré sur le Raspberry Pi.

15h00 :

Une fois que cela est fait, je continue la rédaction de mon rapport.

17h00 :

Fin de séance.

**Date séance : 31 juillet 2013**

Place de travail : King library

08h30 :

Je commence à relire tout mon rapport afin de corriger les éventuelles fautes d'orthographe ainsi que les tournures de phrases que j'ai faites. La relecture du rapport prend beaucoup de temps et demande beaucoup de concentration.

17h00 :

Fin de séance.

**Date séance : 1 août 2013**

Place de travail : King library

08h30 :

Je passe en revue le code des programmes et des scripts que je vais devoir envoyer et je relis aussi quelques parties du rapport et complète quelques détails dans le rapport.

15h00 :

Je fais la mise en page du rapport et regarde que le PDF soit bien en forme avant de l'envoyer.

17h00 :

Fin de séance.