Utiliser le Wifi du NodeMCU

On va commencer par le début. Récupérer le programme exemple en allant *Fichier⇒Exemples⇒ESP8266WiFi⇒WiFiScan*

```
Dans la fonction de configuration setup, on met le module WiFi en mode Station (on aurait pu le mettre aussi en point d'accès). Ne cherchez pas la méthode mode dans la doc Arduino, il s'agit d'une commande spécifique au module ESP (voir ici).  La méthode disconnect permet ensuite de déconnecter le module ESP d'un point d'accès, au cas où !

Dans la fonction boucle (loop) on retrouve les instructions qui permettent d'afficher périodiquement les WiFi captés par le module ESP. La méthode scanNetwork permet , comme son nom l'indique, de scanner les différents canaux (fréquences) dédiés au WiFi et retourner le nombre de réseaux trouvés :

 int n = WiFi.scanNetworks();

Ainsi, si ce nombre n'est pas nul, ils vont être affichés un par un grâce à une boucle for.

   for (int i = 0; i < n; ++i) {
      // Print SSID and RSSI for each network found
      Serial.print(i + 1);
      Serial.print(": ");
      Serial.print(WiFi.SSID(i));
      Serial.print(" (");
      Serial.print(WiFi.RSSI(i));
      Serial.print(")");
      Serial.println((WiFi.encryptionType(i) == ENC_TYPE_NONE)?" ":"*");
      delay(10);
   }

La méthode SSID permet d'afficher le nom du réseau WiFi. Dans le vocabulaire Wi-Fi, SSID veut dire Service Set IDentifier, mais bon c'est pas plus limpide ! La méthode RSSI (Received Signal Strength Indication) affiche la puissance du signal reçu. Les box et plus généralement les points d'accès WiFi émettent 10 fois par seconde un message donnant le nom du réseaux Wi-Fi. La méthode scanNetwork écoute toutes les fréquences et récupère tous ces noms.
```

Une fois connecté au réseau WiFi, nous ne pouvons pas encore communiquer avec d'autres équipements car nous n'avons pas d'adresse IP.

```
Pour obtenir une adresse, nous nous appuyons sur le programme WiFiClient_simple.ino comme son nom l'indique est simple et utilise le WiFi. On retrouve la connexion au réseau WiFi avec la méthode begin :
```

```
  WiFi.begin(ssid, password);    // On se connecte
  while (WiFi.status() != WL_CONNECTED) { // On attend
    delay(500);
    Serial.print(".");
  }
```

La boucle while permet d'attendre que la connexion soit effective et que le serveur DHCP de la box ait fourni les paramètres nécessaires pour se connecter au réseau Internet, qui sont affichés avec le code suivant :

```
  Serial.println("");  // on affiche les paramètres
  Serial.println("WiFi connecté");
  Serial.print("Adresse IP du module EPC: ");
  Serial.println(WiFi.localIP());
  Serial.print("Adresse IP de la box : ");
  Serial.println(WiFi.gatewayIP());
```

La gateway correspond à machine vers laquelle notre module ESP va envoyer les informations, c'est-à-dire à l'adresse IP de la box.

Le modèle client/serveur

Le web est la huitième merveille du monde, au point qu'on le confond souvent avec l'Internet. Il ne s'agit pourtant que d'un des services possible sur le réseau. Ce qui est remarquable avec le web c'est sa scalabilité. C'est à dire la possibilité de répondre à un nombre très important de requêtes sans perdre en performances. Cela est dû à un ensemble de règles pour la conception des services.

Le Web fonctionne sur le mode client/serveur. Le client veut une information et le serveur obéit en la retournant. Si le serveur devait se rappeler de tout ce qu'il a fait, il serait vite débordé. Alors la première règle fondamentale est que seul le client va garder une mémoire (un état) de ce qui se passe. Le serveur va envoyer ce qu'on lui a demandé et va passer à la demande suivante et ainsi de suite. Comme il y a beaucoup plus de clients que de serveurs, c'est plus facile à gérer. Par exemple, quand une page web est demandée à un serveur, elle va contenir d'autres références. Le client reçoit cette information, va la traiter, l'afficher et demander aux serveurs les éléments qui lui manquent. Ces informations peuvent être sur le même serveur ou sur un autre, ce n'est pas important car les serveurs ne se rappellent de rien après avoir traité la requête. On peut ainsi concevoir un service qui repose sur plusieurs serveurs et ainsi répartir le trafic et le traitement.
Les méta-données

Les éléments traités par un serveur sont appelés ressources. La définition de ressource est très vague. Il s'agit d'une information binaire de taille finie. Cela peut correspondre à du texte, à une image, une vidéo, des données,... En plus des données, on peut mettre des méta-données. C'est à dire des choses qui sont au-dessus des données, comme la méta-physique est

au-dessus de la physique ou le métabolisme est au-dessus du bolisme ;).

Les méta-données vont permettre de mieux comprendre les données. Cela peut correspondre à leur date de production, à leur taille ou leur type. Cette dernière information est importante car elle permettra au client de bien les traiter. Dans la réponse d'un serveur, les méta-données (ou l'en-tête) sont séparées des données par une ligne vide, comme l'a montré la réponse qui a été affichée par l'Arduino pour le compteur.

```
HTTP/1.0 200 OK
Date: Thu, 28 Apr 2016 17:51:07 GMT
Server: WSGIServer/0.1 Python/2.7.9
X-Frame-Options: SAMEORIGIN
Content-Type: text/plain

147
```

Dans cet exemple, la première ligne donne le statut de la requête, HTTP/1.0 donne la version du protocole. Le second champ, ici 200 va donner le code indiquant le succès ou l'échec du traitement de la requête par le serveur. On a de la chance, 200 signifie que c'est bon. Le premier chiffre donne la nature de la notification :

```
    1XX : indication d'un traitement en cours
    2XX : succès
    3XX : redirection, il faut interroger un autre serveur pour avoir la
réponse
    4XX : erreur du coté du client
    5XX : erreur du coté du serveur
```

Ainsi la célèbre erreur 404 indiquant que la page n'est pas trouvée est bien une erreur du client qui pose des questions stupides au serveur. La ligne suivante donne la date, à laquelle la page a été produite par le serveur. La ligne suivante, un peu plus énigmatique, elle indique que notre super page web ne peut pas être incluse dans la page web d'un autre serveur.

La dernière ligne de l'en-tête donne la nature de l'information, ici c'est du texte non formaté.

Ensuite on a une ligne vide suivie de ce qui sera affiché, à savoir la valeur du compteur.

Les URI

Un autre facteur clé pour le succès du Web, en plus de l'utilisation de serveurs simples, est l'utilisation d'une désignation uniforme des ressources. Initialement appelées URL (Uniform Resource Locator), on emploie maintenant plus le terme URI (Uniform Resource Identifier) plus générique et qui peut couvrir d'autres technologies (comme la téléphonie sur Internet). Mais de manière simple, une URI se structure de la manière suivante quand on l'utilise pour le Web ou l'Internet des objets :

```
schéma://serveur:port/chemin/jusqu/a/la/ressource

Contrairement à l'idée admise, la première information, schéma, ne
correspond pas au protocole employé sur le réseau, mais indique comment le
reste de l'URI va être structuré. Pour le Web, on retrouve principalement
les schémas pour http (ou https pour sa version sécurisée). Il est suivi par
le nom d'un serveur (ou son adresse IP), un numéro de port peut être
indiqué. Si ce n'est pas le cas, on utilisera le port 80 qui désigne le
protocole HTTP. Ensuite on retrouve un chemin interne au serveur qui permet
de localiser la ressource.

Ainsi cet URI (qui pointe vers un excellent MOOC) :

https://www.fun-mooc.fr/courses/MinesTelecom/04011S02/session02/about

utilise le schéma de représentation https, il y a donc ensuite un serveur
www.fun-mooc.fr  et le chemin est
courses/MinesTelecom/04011S02/session02/about
```

```
Le programme

Le programme Arduino que l'on a fait permet d'ouvrir la connexion vers le
serveur api.tom.tools sur le port 80.

Si on le regarde avec plus de détail le programme WiFiClient-compteur.ino,
dans la boucle (loop) on retrouve  cette instruction :

  // le serveur Web attend traditionnellement sur le port 80
  const int httpPort = 80;

  // Si la connexion a échouée, ça sera pour la prochaine fois
  if (!client.connect(host, httpPort)) {
    Serial.println("connection failed");
    return;
  }

host et httpPort sont deux variables déclarées en début de programme qui
valent respectivement api.tom.tools, soit le nom du serveur et connect
permet d'ouvrir la connexion TCP avec le serveur. Si elle échoue, elle
retourne false qui fait afficher un message d'erreur. Si par contre la
connexion réussit, on continue à dérouler le programme :

  String url = String("/hits/");

On place dans la variable url, le chemin pour accéder à la ressource sur le
serveur. On vient de construire l'URI suivante http://api.tom.tools/hits.

  client.print(String("GET ") + url + " HTTP/1.1\r\n" +
               "Host: " + host + "\r\n" +
               "Connection: close\r\n\r\n");
```

On envoie ensuite sur la connexion TCP une chaîne de caractères comportant 3 lignes (\r\n indique qu'il s'agit d'un changement de ligne). La première ligne contient la requête. Elle commence par l'instruction :

    GET permet de récupérer la valeur de la ressource.
    url contient le chemin vers cette ressource.
    HTTP/1.1 désigne la version du protocole HTTP que l'on souhaite utiliser avec le serveur. Bien entendu, si le serveur ne connaît pas cette version, il répondra avec une autre version. On peut le voir dans la réponse examinée précédemment que le serveur avait répondu avec la version HTTP/1.0.

La deuxième ligne est plus complexe à comprendre. Elle permet de faire de la virtualisation, c'est-à-dire faire tourner plusieurs serveur Web indépendants sur la même machine. Supposons que nous voulions mettre deux serveurs api.tom.tools et www.justinbeiber-fan.fr sur la même machine. Ces deux serveurs vont avoir la même adresse IP. Donc, quand on ouvre une connexion, au niveau IP et TCP, il est impossible de savoir si l'on veut avoir des informations sur le Nelson ou connaître les frasques de notre chanteur favori. En rappelant le nom du serveur après la commande Hosts, le serveur saura quoi répondre.

La dernière ligne indique que l'on peut fermer la connexion quand tout est fini.

Une fois cette requête lancée, on attend une seconde pour avoir la réponse du serveur, puis :

```
  delay(1000);
```

Tant qu'il y a des données reçues, on les affiche ligne par ligne :

```
  while(client.available()){
    String line = client.readStringUntil('\r'); // découpe ligne par ligne
    Serial.print(line);
  }
```

et on ferme la connexion TCP. On attend 30 secondes et on recommence en ouvrant une connexion...

```
  Serial.println();
  Serial.println("connexion fermée");
```

Intéressons nous à la température

En reprenant la structure du programme d'interrogation du compteur, il est facile de la transformer pour interroger un autre serveur. Intéressons-nous maintenant au programme WiFiTemperature.ino qui va interroger le serveur openweathermap.org :

```
// valeurs pour le serveur Web
const char* host     = "api.openweathermap.org";
```

```
const char* apikey   = "votre clé API (apikey)"; // il est possible
d'utiliser la clé d'API suivante : 1a702a15a2f46e405e61804cf67c0d30
const char* town     = "Rennes,fr";
```

On ajoute aussi deux variables apikey et town qui serviront à construire l'URL.

```
  String url = String("/data/2.5/weather?q=") + town + "&appid=" + apikey;
```

Le chemin est un peu plus complexe car il contient une partie fixe et à la fin il y a un point d'interrogation. Formellement, il s'agit de paramètres que l'on fournit à la ressource /data/2.5/weather pour qu'elle puisse fournir un résultat. Ici, on indique la ville et l'API key qui sert à identifier l'utilisateur qui fait la requête.

On attend la réponse:

```
  // On attend 10 millisecondes
  delay(10);
```

puis on ignore toutes les méta-données de l'en-tête en attendant une ligne vide :

```
  inBody = false; // on est dans l'en-tête
  // On lit les données reçues, s'il y en a
  while(client.available()){
    String line = client.readStringUntil('\r');
    if (line.length() == 1) inBody = true; /* passer l'en-tête jusqu'à une
ligne vide */
    if (inBody) {                          // ligne du corps du message, on
cherche le mot clé
```

La variable booléenne inBody reste à faux tant que l'on ne reçoit pas une ligne vide, c'est-à-dire d'une longueur de 1 (en prenant en compte le retour à la ligne). Si on est dans le corps (c'est-à-dire les données) on peut rechercher le mot clé qui nous intéresse dans la structure. Il est indiqué au début du programme :

```
 String keyword = String("\"temp\":");  //chaîne que l'on recherche dans le
JSON
```

Notez les \ qui permettent de mettre les " dans la chaîne de caractères, donc en fait on recherche "temp" :

```
    if (inBody) {  // ligne du corps du message, on cherche le mot clé
       int pos = line.indexOf(keyword);

      if (pos > 0) { /* mot clé trouvé */
       // indexOf donne la position du début du mot clé, en ajoutant sa
longueur
```

```
      // on se place à la fin.
      pos += keyword.length();
```

Si pos est positif, c'est que le mot clé a été trouvé dans la ligne. Comme pos donne l'endroit où cette chaîne de caractères commence dans la ligne, en additionnant sa taille, on obtient la fin de la chaîne de caractères, c'est-à-dire le début du nombre recherché. Comme c'est également une chaîne de caractères, pour le transformer en nombre flottant (c'est-à-dire à virgule) on utilise la commande atof.>

Cette commande prend l'adresse en mémoire du début de la chaîne de caractères, c'est pour cela que l'opérateur & est utilisé :

```
      temperature = atof(&line[pos]);
```

Voilà on a la valeur, il ne vous reste plus qu'à la combiner avec la commande vers le servo de votre Nelson pour le transformer en thermomètre !
LE JSON

JSON - prononcé jisone en français ou jay-zon avec l'accent anglais - n'est pas un personnage de films d'horreur, mais l'acronyme de Java Script Object Notation. C'est un format qui a été à l'origine conçu pour transporter des données pour les pages Web. Mais, à l'inverse du tueur en série, JSON étant très tolérant, il a été très vite utilisé pour structurer d'autres informations et est très populaire pour l'internet des objets, car, en plus, il est compact et simple à utiliser.

Structurer des données est quelque chose de très simple : sionnestructurepasunephraseenmettantdesespacesilesttrèsdurdelalireetdyretrouverdesmots.
Pour les données c'est la même chose. Si on veut uniquement avoir une température on peut très bien retourner juste 25, mais si dans la réponse on veut mettre d'autres informations, il faut définir une structure. Cela pourrait être par exemple des espaces séparant les valeurs, mais très vite il est difficile de se rappeler à quoi correspond telle colonne.

JSON définit deux types de données de base, les chaînes de caractères qui sont entre " et les nombres qui sont non pas entre " et qui contiennent principalement des chiffres. Par exemple "123" est une chaîne de caractères et -.1e+3 est un nombre (c'est -100). Pour ce dernier, on a un peu exagéré en utilisant la notation eX qui correspond à 10 puissance X (comme sur les calculatrices).

Ensuite on va avoir deux types structures, les listes de paires qui se composent de deux champs, le mot clé qui est un chaîne de caractères, suivi de : et d'une valeur qui peut être n'importe quoi. Les listes sont délimitées par des accolades. Par exemple, nous pourrions donner donne les propriétés d'un écran avec le json suivant :

```
{ "resolution_verticale": 1200, "resolution_horizontale": 900, "couleurs": true}
```

Json permet également de faire des tableaux :

```
[1200, 900, true]
```

On peut noter que c'est beaucoup plus compact, mais dans ce cas la position est importante, il faut savoir à quoi correspond le premier élément, le second... Un tableau est délimité par des crochets droits et les éléments sont séparés par des virgules. La beauté de JSON est que l'on peut faire des liste de tableaux, des tableaux de listes, des tableaux de tableaux, des listes de listes, de tableaux de listes de tableaux de listes... Par exemple :

```
 {
    "Image": {
        "Hauteur":  800,
        "Largeur": 600,
        "Titre":  "Vue du 5ieme étage",
        "Vignette": {
            "Url":    "http://www.example.com/image/481989943",
            "Hauteur": 125,
            "Largeur":  100
        },
        "Animee" : false,
        "IDs": [116, 943, 234, 38793]
      }
  }
```

Ce JSON décrit l'image par une liste de paires, donnant la hauteur, la largeur, un titre, une sous-liste qui décrivent une vignette,... Si vous lisez l'anglais, vous pouvez regarder la description officielle de JSON dans un standard.