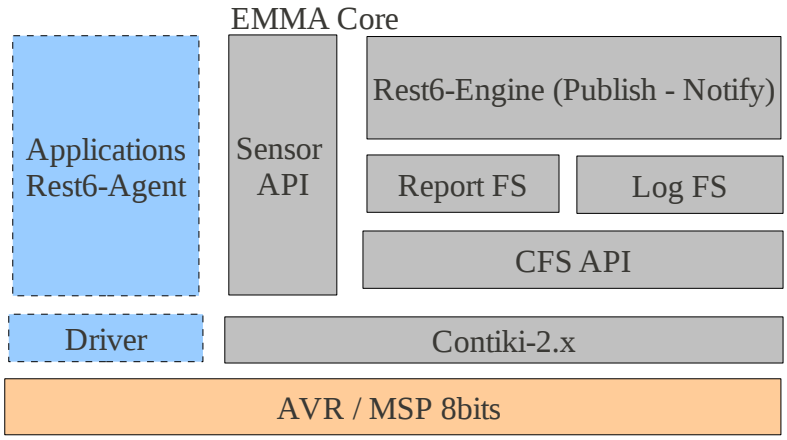
E.M.M.A

Energy Monitoring Management Agent



Key words :

- **EMMA**
- **JSON RestFull Engine**
- **TCP/ IPV6**
- **6loWPAN**
- **IEEE 802.15.4**
- **Contiki-2.x**

Emma Network

Le principe fondamentale d'Emma Network est de répartir les tâches et les responsabilités sur l'ensemble du réseau. Les noeuds communiquement en paire à paire directement au sein du LowPAN. Un edge router réalise la passerelle et permet l'auto-établissement du réseau au niveau des couches **802.15.4 et 6loWPAN** (IPV6 Router Advertissement).

L'ensemble du réseau (LowPAN et Over Internet) fonctionne sur le protocole HTTP en RestFull JSON. Chaque noeud embarque un client-serveur web permettant des requêtes bidirectionnelles sans modifcation du protocole HTTP ni l'emploi d'un service annexe.

Pour éviter une concentration permanente de la prise de décision sur un noeud, chaque noeud peut demander des reports par notification ou transmettre des ordres par notification. Pour cela un mécanisme de configuration du réseau à été mis en place. Un report est posté sur un noeud pour lui déléguer une tâche ou une responsabilité, ce report est entièrement configurable. (host, port, uri, body, min, max period, etc...)

Finalement, l'utilisation du réseau ne nécessite pas la présence d'un serveur permanent de concentration des données et de prise de décision mais d'un serveur de monitoring et de management du réseau, celui s'occupe d'attribuer les report et order par notification au noeud et vérifie que ceux-ci suivent correctement les ordres. La feedback remontait par le proxy ou par les noeuds directement permet au manager de réseau d'effectuer les ajustement de configuration du réseau.

Le **Network Manager** établit des routes de décision à travers les noeuds du réseau, vérifie le bon déroulement des tâches définies et réajuste la configuration des reports si nécessaire. Un Network Manager n'appartient pas forcément au réseau LowPAN mais peut être **un serveur multi-site** (à travers un tunnel 6to4) puisque sa présence n'est pas nécessairement permanente. Nous pouvons imaginer un Network Manager présent sur un Smart Phone dans le cas d'un réseau EmmaNetwork statique.

Si le Network Manager est en permanence connecté et monitoring & manage en continue, le réseau est dit **« dynamique »** puisque celui-ci s'auto-reconfigure les routes de décision pour optimiser une fonction ou corpus de fonctions spécifiques définit dans le Network Manager. Dans le cas, d'un Network Manager ponctuellement présent, le réseau est dit « **statique »**.

Le mécanisme de report peut également réaliser des tâches plus complexes puisqu'il est à même de se connecter à n'importe quelles ressources sur Internet. Cependant pour des raisons d'optimisation des flux réseaux au niveau de la passerelle, il est conseillé que les « **Web Intermediaries »** accèdent au proxy sur Internet pour récupérer les datas des noeuds LowPAN et inversement, les noeuds LowPAN accèdent au proxy pour récupérer des données issues de ressources Internet.

Le principal intérêt d'employer une couche HTTP RestFull JSON pour communiquer avec les noeuds LowPAN est de pouvoir délocaliser le Network Manager sur Internet et de définir une couche de communication homogène sur l'ensemble du système.

Scénarios : Cas d'utilisation dans un couloir

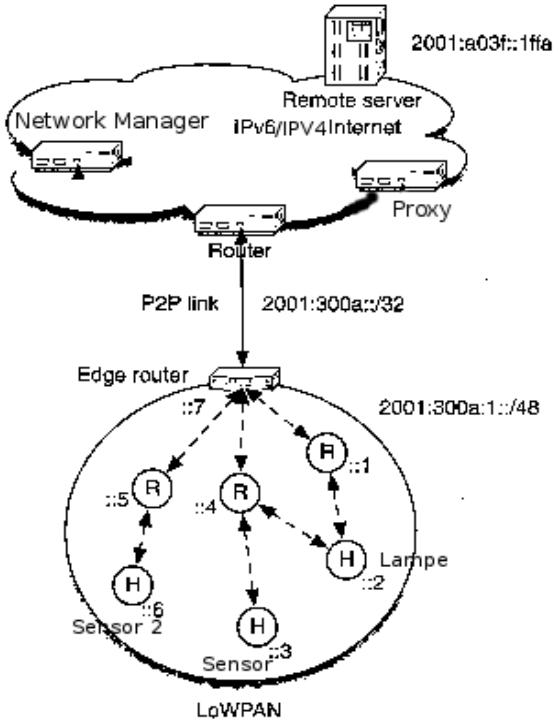


Figure 1.13 A 6LoWPAN example.

- EmmaNetwork Statique

Supposons un réseau LoWPAN dans le couloir. Il se compose d'une lampe et de deux capteurs situés de part et d'autres du couloirs indiquant l'arrivée d'une personne et l'ensoleillement.

Configuration typique :

- Sensor1->Proxy sur /data/*/report/
  POST http://[Proxy]:8081/uri1/)

- Sensor2->Proxy sur /data/*/report/
  POST http://[Proxy]:8081/uri1/)

- Lampe->Proxy sur /data/*/report/
  POST http://[Proxy]:8081/uri1/)

- Sensor1->Lampe sur /data/presence/report/ et /data/brightness/report/
  PUT http://[Proxy]:8081/data/white1/ {« value »: « 1000 »}

- Sensor2->Lampe sur /data/presence/report/ et /data/brightness/report/
  PUT http://[Proxy]:8081/data/white1/ {« value »: « 1000 »}

Lors d'un changement d'état d'une ressource des capteurs ou de la lampe, le proxy reçoit une notification HTTP avec les nouvelles valeurs des ressources. La lampe reçoit en même temps un ordre d'allumage ou d'extinction.

Les **reports/order par notification HTTP** sont actuellement déclenchés par le changement

d'état d'une ou des ressources cibles à travers les paramètres « min », « max », « period », prochainement un champs «cmd» permettra d'établir des conditions de notifications plus avancées.

Exemple :


```
{
"uri":"http://[3ffe:0501:ffff:0100:0206:98ff:fe00:0231]/notify.php",
"id":"2",
"port":"80",
"method":"GET",
"body":"",
"name":"*",
"period":"100",
"cmd":"( /data/ressource1 > 100 && /data/ressource2 < 1000 ) || http://[Node2]/data/ressource0/log/{currentTime}/ <
{\"InstantaneousValue\": \"600\"}",
"min":"20",
"max":"-1"
}
```


- EmmaNetwork Dynamique

Le réseau est supervisé par des serveurs distants : un proxy s'occupant de bufferiser et de logguer l'exploitation du réseau et d'un Network Manager.

Le réseau peut être **statique permanent ou temporairement** en l'abscence totale ou partiel(Passerelle zombie) du Manager de réseau.

Dans cette configuration les sensors pilotent directement la lampe garantissant ainsi le bon déroulement de l'opération et ceux malgré une éventuelle congestion du trafic au niveau de la passerelle (qui ne fait que relayer les états du réseaux et non les ordres).

Supposons qu'un capteur de luminosité soit défaillant, empêchant ainsi la détection de l'abscence d'éclairage alors que le capteur de présence détecte bien une personne. Le Network Manager reçoit des reports contradictoires avec un capteur qui voit le couloir dans le noir et l'autre le voit éclairer. Il peut alors décider de déléguer la ressource « capteur de lumière » du Sensor1 à la ressource « capteur de lumière » du Sensor2 et notifie l'administrateur du réseau de la défaillance d'un capteur.

La nouvelle auto-configuration deviendrait alors :

- Sensor1->Proxy sur /data/*/report/
  POST http://[Proxy]:8081/uri1/
- Sensor2->Proxy sur /data/*/report/
  POST http://[Proxy]:8081/uri1/
- Lampe->Proxy sur /data/*/report/
  POST http://[Proxy]:8081/uri1/
- Sensor1->Lampe sur /data/presence/report/
  PUT http://[Proxy]:8081/data/white1/ {« value »: « 1000 »}
- Sensor2->Lampe sur /data/presence/report/ et /data/brightness/report/
  PUT http://[Proxy]:8081/data/white1/ {« value »: « 1000 »}
- **Sensor2->Sensor1 sur /data/brightness/report/**
  PUT http://[Sensor1]/data/brightness/ {« value »: «brightness»}

L'exemple ci-dessus décrit une configuration **EmmaNetwork Dynamique** à travers l'auto-configuration du réseau par un Network Manager.

HTTP RestFull Engine Interface :

- **Description globale**

URI et des méthodes

GET	/
GET | PUT	/data/{* | {RessourceName}}/
GET | POST	/data/{* | {RessourceName}}/report/
GET | PUT | DELETE	/data/{* | {RessourceName}}/report/{* | {ReportId}}
GET	/data/{* | {RessourceName}}/log/
GET	/data/{* | {RessourceName}}/log/{* | {LogId}}

Représentation d'une ressource :

```
{
 "name" : "string",
 "unit" : "string",
 "InstatenousRate" : "integer"
}
```

Codes erreurs selon la RFC 2616:

```
200 OK
400 Bad Request
403 Forbidden
404 Not Found
405 Method Not Allowed
408 Request Timeout
411 Length Required
413 Request Entity Too Long
414 Request URI Too Long
500 Internal Error
```

- **Description détaillée :**

Retourne la liste des ressources disponible sur le noeud

```
GET /
Response :
[
{
"name":"ressource",
"uri":"http://[host ::ipv6]/data/{name}/"
},
[...]
]
```

Retourne ou mets à jour l'état instantanné de la ressource

```
GET /data/{* | {RessourceName}}/
Response :
{
"name":"brightness",
"unit":"lux",
"InstatenousRate":"2"
}

PUT /data/{* | {RessourceName}}/
Body :
{
"value":"1000"
}
```

Retourne un log spécifique pour une ressource, ou tous les logs pour une ou toute les ressources.

```
GET /data/{* | {RessourceName}}/log/{* | {LogId}}
Response :
[
{
"id":"192",
"name":"temperature",
"value":"30",
"time":"165"
}
[...]
]
```

Insertion d'une nouvelle demande de report par notification HTTP sur une ou toutes les ressources.

```
POST /data/{* | {RessourceName}}/report/
Body:
{
"host":"[3ffe:0501:ffff:0100:0206:98ff:fe00:0231]",
"uri":"notify.php",
"port":"80",
"method":"GET",
"body":"",
"period":"100",
"min":"30",
"max":"150"
}
```

Note : Seule l'uri et l'host sont des paramètres obligatoires.

Retourne la liste des demandes (ou une demande) de report par notification HTTP

```
GET /data/{* | {RessourceName}}/report/{* | {LogId}}
Response :
[
```

```
{
"uri":"http://[3ffe:0501:ffff:0100:0206:98ff:fe00:0231]/notify.php",
"id":"2",
"port":"80",
"method":"GET",
"body":"",
"name":"*",
"period":"100",
"min":"30",
"max":"150"
},
[...]
]
```

Mise à jour d'une demande de report par notification HTTP.

```
PUT /data/{* | {RessourceName}}/report/{LogId}
Body:
{
"uri":"http://[3ffe:0501:ffff:0100:0206:98ff:fe00:0231]/notify.php",
"id":"2",
"port":"80",
"method":"GET",
"body":"",
"name":"*",
"period":"100",
"min":"30",
"max":"150"
}
```

Note : Tous les paramètres du corps sont optionnels

Suppression d'une ou de toutes les demandes de report par notification HTTP

```
DELETE /data/{* | {RessourceName}}/report/{* | {LogId}}
```

API Emma

- **Prototype**

RESSOURCE_INIT(sensor)
Déclaration d'une ressource

RESSOURCE_UNIT(sensor, unite)
Configuration de l'unité de mesure

RESSOURCE_SET(sensor, setFunc)
Association d'une fonction d'édition à la ressource

RESSOURCE_GET(sensor, getFunc)
Association d'une fonction de lecture à la ressource

RESSOURCE_LOAD(sensor)
Chargement de la ressource dans le moteur de ressource Rest6Engine

RESSOURCE_LIST(name)
Création d'une liste de ressources

RESSOURCE_LIST_LOAD(name)
Chargement d'une liste de ressources externes

- **Example : Application capteurs et Registre data**

Cette application se compose de trois ressources déclarées : deux ressources en lectures seules et une ressource en lecture et écriture.

/contiki-2.x/plateform/avr-meshnetics/Rest6-Agent.c

```
#include "ressources.h"
#include "Rest6-XML.h"

SENSOR_LIST_LOAD(sensors_list);
RESSOURCE_INIT(temperature);
RESSOURCE_INIT(brightness);
RESSOURCE_INIT(data);

int getLight(){
[...]
return light;
}

int getTemperature(){
[...]
return light;
}

void setData(int val){
[...]
}
```

```
int getData(){
[…]
return data;
}

void ressources_loader(){
        RESSOURCE_CONFIGURE(brightness,lux);
        RESSOURCE_CONFIGURE(temperature,celcius);
        RESSOURCE_CONFIGURE(data,%) ;

        RESSOURCE_GET(brightness, &getLight);
        RESSOURCE_GET(temperature, &getTemperature);
        RESSOURCE_SET(data, &setData) ;
        RESSOURCE_GET(data, &getData);
}
```

L'interface entre la couche driver et le core d'Emma-Rest6 se réalise à travers un jeux de pointeurs de fonctions.

Note : Seul les ressources de types entier ou flottant sont gérées.

Fichier par defaut : Si aucun fichier Rest6-Agent.c n'est trouvé lors de la compilation dans le dossier /contiki-2.4/plateforme/, le fichier par défaut chargé est :

/contiki-2.x/apps/Emma-Rest6/Rest6-Agent.c

```
#include "ressources.h"
#include "Rest6-XML.h"
void ressources_loader(){ ;}
```

Emma-Rest6 Engine :

Emma-Rest6 Engine est un moteur de publication, de log et de report RestFull sur IPV6. Il offre une API simple permettant la déclaration de nouvelle ressource et l'association aux couches driver dans un fichier */contiki-2.x/platform/{plateform}/Rest6-Agent.c* sans aucune autre modification.

Type of memory	Size
RAM	1,2 Ko
ROM	15 Ko

Configuration :

/contiki-2.x/apps/Emma-Rest6/Emma-conf.h

```
// Debug mode to print the different step/state of the machine
#define DEBUG 1
// HTTP Notification mode is a light web client to send simple request to another (light) web server.
#define SEND_NOTIFY 1

// Port for listenning incomming request
#define PORT 80

// Periodic time for calling the getter function of each ressource
#define PROCESS_TIME 5
// Periodic time for calling the log process of all ressources
#define PROCESS_TIME_LOG 10
// Periodic time for calling the report process for sending report with HTTP notification
#define PROCESS_TIME_REPORT 10

// Size max of a ressource name
#define RESSOURCE_NAME_SIZE 15
// Length of the max id number
#define RESSOURCE_ID_MAX_DIGIT 3

// Size of the outcomming body request ({\n\"value\":\"xxxx\"\n} = 23)
#define BODY_NOTIFY 25

// Notification buffer size define
// Max size of URI for sending request (/data/{ressource}/{report | log}/{id} = 15 + sizeof({ressource}) +
sizeof({id}))
#define URI_OUTCOMMING_MAX_SIZE (15 + RESSOURCE_NAME_SIZE + RESSOURCE_ID_MAX_DIGIT)
#define URI_INCOMMING_MAX_SIZE (15 + RESSOURCE_NAME_SIZE + RESSOURCE_ID_MAX_DIGIT)


// There are two buffer of size BUFFER_SIZE
#define BUFFER_SIZE 70
#define BUFFER_SOCKET_SIZE 70
#define BUFFERMINI_SIZE 6
#define INCOMMING_CONNECTION_MAX 1
#define OUTCOMMING_CONNECTION_MAX 1
#define TIMEOUT 20
```

API CFS

API CFS est une couche d'abstraction pour allouer des slots de données dans la mémoire EEPROM. Elle alloue des zones EEPROM fixes pour logger les différentes ressources.

Pour protéger l'accés aux ressources dans le cadre d'application multi-tâches, le système de fichier n'a aucune variable propre d'allouée. Le consommateur ou le producteur d'une ressource fournit les buffers nécessaires pour l'écriture, la lecture ou la suppression en mémoire. Ceci permettant une protection d'accés au ressource par contexte en appuie avec des sémaphores.

API CFS Log

```
CFS_LOG_COUNT 20
LOG_NEW(struct Repor * data, char * buf, struct File* cfs)
LOG_UPDATE(struct Repor * data, char * buf, struct File* cfs)
LOG_READ(struct Repor * data, char * buf, struct File* cfs)
LOG_OPEN(struct Repor * data, char * buf, struct File* cfs)
LOG_FORMAT(struct Repor * data, char * buf, struct File* cfs)
LOG_DELETE(struct Repor * data, char * buf, struct File* cfs)
```

La couche applicative de Emma-Rest6 utilise CFS Log en mode circulaire, les logs précédents sont écrasés un à un avec toute fois une variable time croissante. Cette variable correspond au nombres de secondes entre la création du log et la mise en service du noeud.

Log Object

```
{
"id":"192",
"name":"temperature",
"value":"30",
"time":"165"
}
```

Location	Memory
Report memory size (per unit)	40 octets
Report transport size (per unit)	59 octets

API CFS Report

```
CFS_REPORT_COUNT 4

REPORT_NEW(struct Repor * data, char * buf)
REPORT_UPDATE(struct Repor * data, char * buf)
REPORT_READ(struct Repor * out, char * buf)
REPORT_OPEN(struct Repor * out, char * buf)
REPORT_FORMAT(char * buf, char * buff2)
REPORT_DELETE(char * buf, char *buff2, id)
```

Report Object

```
{
"uri":"http://[3ffe:0501:ffff:0100:0206:98ff:fe00:0231]/notify.php",
"id":"2",
"port":"80",
"method":"GET",
"body":"",
"name":"*",
"period":"100",
"min":"30",
"max":"150"
}
```

Location	Memory
Report memory size	92 octets
Report transport size	164 octets

Contraintes :

Mémoire : Deux buffers principaux sont employés pour chaque connection, la taille des buffers a un impacte immédiat sur la vitesse de traitement puisqu'un algorithme auto-fragmente les requêtes JSON selon la taille des buffers disponibles.

Temps : Les timer pour les process report, log et driver ne doivent pas être trop rapprochés, l'ordonnancement sera toujours réalisé, ils ralentiront significativement la réactivité du noeud. La planification du process de la stack réseau UIP serait alors moins fréquente augmentant ainsi le temps de traitement général.

La configuration suivante par défaut est optimisé pour des microcontroleur cadencé à 8 Mhz :

```
// Periodic time for calling the getter function of each ressource
#define PROCESS_TIME 5
// Periodic time for calling the log process of all ressources
#define PROCESS_TIME_LOG 10
// Periodic time for calling the report process for sending report with HTTP notification
#define PROCESS_TIME_REPORT 10
```

Requêtes : Les connections simultannées sont limitées par la taille de la mémoire RAM et sont fixéxes lors de la compilation pour protéger le noeud des dépassements mémoires ou des erreurs d'allocations dynamiques.

Type de connection	RAM
Publication entrant	276 octets
Notification sortant	262 octets