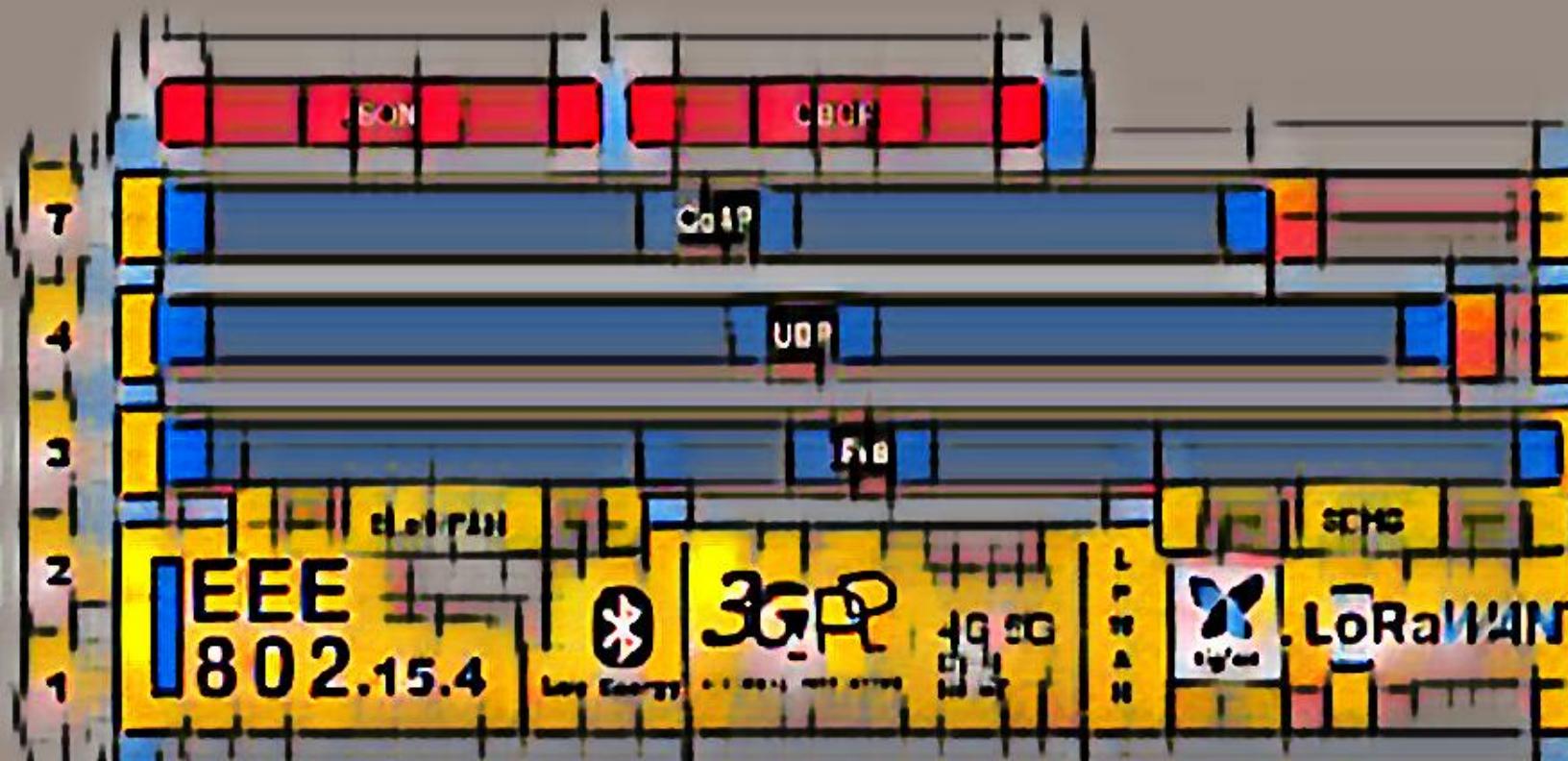


PROGRAMMER L'INTERNET DES OBJETS

Laurent TOUTAIN



IMT ATLANTIQUE

Ce document est mis à disposition selon les termes de la licence Creative Commons “Attribution – Pas d'utilisation commerciale – Pas de modification 3.0 non transposé”.



Basé sur le MOOC PLIDO.

Publié le 18 août 2022



Table des Matières

0.1	Ressources disponibles	13
0.2	Auteurs	13
1	LES BASES DE L'INTERNET DES OBJETS (IOT)	16
1.1	Introduction	16
1.1.1	Réseaux dédiés	16
1.1.2	3 phases technologiques	17
1.2	L'Internet des Objets	17
1.3	Le problème	19
1.4	Evolution de l'IoT	20
1.5	Des objets contraints	21
1.6	L'interopérabilité	22
1.7	Le besoin de standardisation	24
1.8	Questions	25
2	ARCHITECTURE DE L'INTERNET	26
2.1	Les protocoles	26
2.2	Les fondements du Web	29
2.2.1	Ressources	30
2.2.2	Identifiants	30
2.2.3	Interactions	33

2.3	Modèle Publish/Subscribe	34
2.3.1	MQTT	35
2.3.2	différence avec REST	36
3	Wireshark	37
3.1	Installation	37
3.2	Démarrage	37
3.3	Capture	39
3.3.1	Analyse du trafic web	41
3.3.2	Analyse des requêtes HTTP	43
3.3.3	Analyse de la pile protocolaire	45
3.4	Do it yourself	48
4	Modbus	50
4.1	Introduction	50
4.1.1	Registres	51
4.1.2	Protocole	51
4.1.3	Exemple : XY-MD02	51
4.1.4	Passerelle IP	54
5	ARCHITECTURE POUR L'IOT	59
5.1	Introduction	59
5.2	Topologies	59
5.3	Niveaux 1 et 2	61
5.4	IP et couches d'adaptation	62
5.5	Mise en œuvre de REST	63
5.6	Représentation des données	63
5.7	Alternatives à REST	63
6	LA REPRESENTATION DE LA DONNEE	64
6.1	Introduction	64
6.2	La sérialisation	65
6.3	Base64	67
6.4	HTML	69
6.5	XML	69
6.6	JSON	70

6.7	CBOR	73
6.7.1	CBOR en python	74
6.7.2	Type Entier Positif	74
6.7.3	Type Entier Négatif	76
6.7.4	Type Séquence binaire ou Chaîne de caractères	77
6.7.5	Type tableau	78
6.7.6	Type Map (Liste de paires)	78
6.7.7	Type Map (List of pairs)	78
6.7.8	Type étiquette	79
6.7.9	Le type flottant et valeurs particulières	81
6.8	Questions sur CBOR	81
6.9	SenML	82
7	Mettre en œuvre un Capteur Virtuel	84
7.1	JSON	84
7.1.1	Serveur Minimal	84
7.1.2	Capteur virtuel	85
7.2	CBOR	88
8	Séries temporelles	91
8.1	Envoi d'un tableau	91
8.2	Codage par différence	92
8.3	Architecture	93
8.4	Beebotte	94
8.4.1	Configuration	94
8.4.2	Enregistrement des ressources	94
8.4.3	Visualisation des ressources	97
8.5	Interopérabilité	98
8.6	et SenML ?	98
9	Découvrons le LoPy	104
9.1	Introduction	104
9.2	Installation d'Atom	105
9.2.1	Communiquez avec votre Pycom	106
9.2.2	Installez votre environnement de travail	107
9.3	Connexion au réseau Wi-Fi	108
9.4	Mise en place d'un client	109
9.5	BME 280	110
9.5.1	Le bus I2C	110

9.5.2	Mesure de la température	112
9.6	Thermomètre Wi-Fi	114
10	Sigfox	116
10.1	Récupération des identifiants	116
10.2	Enregistrement de l'objet	117
10.3	Visualisation des données émises par le Pycom	117
10.4	Que s'est-il passé coté radio	118
10.5	Récupération des données	118
10.5.1	Sur le serveur	120
10.5.2	Sur le LoPy	122
10.5.3	requête GET depuis le serveur	124
10.5.4	requête POST vers le serveur	126
10.6	Conclusion	130
11	LoRaWAN	131
11.1	Information sur le LoPy	132
11.2	The Things Network	132
11.3	Ajout d'une passerelle radio	147
11.3.1	Installation du Pygate	147
11.3.2	Configuration de The Things Network	148
11.4	Vue générale des échanges	148
11.5	Thermomètre LoRaWAN	149
12	CoAP	151
12.1	Introduction	151
12.2	Format d'une en-tête CoAP	152
12.2.1	Codage de code	153
12.2.2	Utilisation du champ Message ID	153
12.2.3	Les Token	156
12.2.4	Les options CoAP	158
12.2.5	Options CoAP	160
12.2.6	Représentation des URI	160
12.2.7	Représentation des données	163
12.3	Observe	164

13	Experimentons CoAP	167
13.1	Mise en œuvre du client/serveur	167
13.1.1	aiocoap	167
13.1.2	côté Objet	169
13.2	GET /time	171
13.3	POST	175
13.3.1	Ressource codée en ASCII	175
13.3.2	Ressource codée en CBOR	177
13.3.3	No Response	180
13.4	Chaîne complète de remonté de mesures	180
13.5	SCHC	181
13.5.1	Emission côté client	182
13.5.2	Réception côté serveur	183
13.5.3	serveur CoAP	184
13.6	Pistes d'améliorations	184
14	LwM2M	185
14.1	Introduction	185
14.2	Architecture	186
14.3	Enregistrement d'un Objet	187
14.3.1	Analyse de l'en-tête CoAP	187
14.3.2	Analyse du contenu du POST	189
14.3.3	Définition des ressources	190
14.4	Resource Directory	195
14.5	interrogation du client LwM2M	196
14.5.1	Lecture simple	198
14.5.2	Lecture d'une instance	199
14.5.3	Observe	199
15	Réponses aux questions	201

Acronyms

3GPP 3rd Generation Partnership Project	IP Internet Protocol
ABP Authentication By Personalisation	IPv4 Internet Protocol version 4
ADSL Asymmetric Digital Subscriber Line	IPv6 Internet Protocol version 6
AMQP Advanced Message Queuing Protocol	IPSO IP for Smart Objects
AS Application Server	ITU International Telecommunication Union
ASCII American Standard Code for Information Interchange	IRI International Resource Identifier
BLE Bluetooth Low Energy	ISBN International Standard Book Number
CBOR Concise Binaire Object Representation	ISO International Standardization Organization
CoAP Constrained Application Protocol	JMS Java Messaging Service
Cosem Companion Specification for Energy Management	JSON JavaScript Object Notation
CRC Cyclic Redundancy Check	JSON-LD JavaScript Object Notation for Linked Data
CSV Comma Separated Values	LCIM Levels of Conceptual Interoperability Model
DLMS Device Language Message Specification	LPWAN Low Power Wide Area Network
DTT Digital Terrestrial Television	LwM2M Lightweight Machine to Machine
DR Data Rate	LNS LoRaWAN Network Server
GSMA GSM Association	MQTT Message Queuing Telemetry Transport
HTML HyperText Markup Language	NAT Network Address Translation
HTTP HyperText Transport Protocol	NGW Network GateWay
HTTPS HyperText Transport Protocol Secure	NIDD Non IP Data Delivery
IANA Internet Assigned Numbers Authority	OMA Open Mobile Alliance
IBAN International Bank Account Number	OTAA Over The Air Authentication
IEEE Institute of Electrical and Electronics Engineers	OVH On Vous Herbègue
IETF Internet Engineering Task Force	PAC Porting Authorization Code
IoT Internet of Things	REST REpresentational State Transfer
	RFC Request For Comments

RGW	Radio GateWay	TNT	Télévision Numérique Terrestre
RNIPP	Répertoire National d'Identification des Personnes Physiques	TTN	The Things Network
RSSI	Received Signal Strength Indicator	UDP	User Datagram Protocol
RTT	Round Trip Time	UIT	Union internationale des télécommunications
SCEF	Service Capability Exposure Function	UNB	Ultra Narrow-Band
SenML	Sensor Measuring List	URI	Universal Resource Identifier
SCHC	Static Context Header Compression	URL	Universal Resource Locator
SF	Spreading Factor	URN	Universal Resource Name
SNR	Signal to Noise Ratio	VPS	Virtual Private Server
SSID	Service Set IDentifier	W3C	World Wide Web Consortium
STIC	Sciences et Technologies de l'Information et de la Communication	WWW	World Wide Web
TCP	Transmission Control Protocol	XML	Extensible Markup Language
TLV	Type Length Value	XMPP	Extensible Messaging Protocol et Presence



Introduction

L'internet des Objets ne va pas seulement ajouter une nouvelle catégorie d'équipements au réseau, il va également modifier la mise en œuvre des protocoles. Il faut donc être à la fois différent mais identique, disruptif mais conservateur. L'Internet des Objets constitue une évolution majeure des protocoles mondiaux pour répondre à deux défis fondamentaux : être économique en énergie et surtout être interopérable ; c'est-à-dire permettre aux Objets de s'intégrer facilement dans les systèmes d'information existants et au final l'information produite banalisée.

L'IoT modifie également la manière d'enseigner. Jusqu'à présent cet enseignement était très stratifié, on montrait en cours magistral comment fonctionne les protocoles, comment ils s'organisent. La mise en œuvre pratique portait plutôt sur la configuration des équipements d'interconnexion comme les routeurs. La vision traditionnelle consiste à bien séparer les fonctionnalités. Les protocoles sont empilés les uns sur les autres avec les fonctionnalités comme le codage de l'information, sur lesquelles reposent l'aiguillage de l'information et au sommet les applications. Chaque protocole dans cet empilement indépendant les uns des autres et des frontières très strictes.

Cette vision est à revoir pour l'Internet des objets, les faibles capacités en mémoire, la nature des liens de communication font qu'il est difficile de se spécialiser dans un seul domaine. Ainsi, j'espérais avoir oublié pour toujours les aspects traitement du signal ou électronique en quittant l'université. Or pour concevoir un objet une approche pluridisciplinaire est essentielle, il faut donc à la fois appréhender des concepts électroniques aussi bien en terme de place que de consommation électrique, de traitement du signal car les signaux sont généralement très faibles, sans oublier les problèmes traditionnels en réseaux comme le routage, l'auto-configuration ou la sécurité. Il faut également avoir un œil sur les applicatifs : comment les données sont représentées et codées lors de leur transmission et comment elles peuvent interagir avec des services existants.

L'Objet en lui-même n'est qu'une petite partie du problème, il va envoyer un flux d'information plus ou moins important vers des serveurs qui seront chargés de les analyser, de les stocker, de trouver des tendances. Les petits ruisseaux faisant les grandes rivières, il faut que ces serveurs ou les réseaux qui y conduisent soient correctement dimensionnés. Mais également que le coût pour la gestion d'un objet soit très faible sinon en les cumulant sur des millions d'objets il peuvent être un

frein au déploiement.

Ce livre est le fruit des expériences que nous avons eu avec les ateliers de fabrication, et d'une constatation que le réseau est souvent le parent pauvre à la fois des applications créées mais également du support que l'on trouve sur ces plates-formes ; les communications sont vues dans une optique de l'application, sans prendre en compte une vision d'interopérabilité plus globale conduisant au concept d'Internet des Objets. Ceci peut se comprendre car les piles protocolaires de l'Internet sont relativement importantes et dans un environnement restreint, il est plus facile de s'en débarrasser. Néanmoins, ces piles protocolaires ont un avantage, elles favorisent la communication entre les constituants du réseau. Ainsi, il est possible de piloter un objet à partir de son ordinateur portable, les données peuvent être envoyées sur des serveurs pour être traitées,...

La richesse des possibilités de communication va permettre de créer des services innovants. Il faut aussi que les piles protocolaires s'adaptent. C'est ce que font de nombreux organismes dont l'IETF qui standardise les protocoles utilisés dans l'Internet.

Cet ouvrage est une adaptation du MOOC Programmer L'Internet des Objets sur Coursera. Il va couvrir les technologies, architectures et protocoles nécessaires pour la réalisation de bout en bout de la collecte d'informations sur des réseaux dédiés à l'IoT à la structuration de la donnée et à son traitement.

Vous allez notamment :

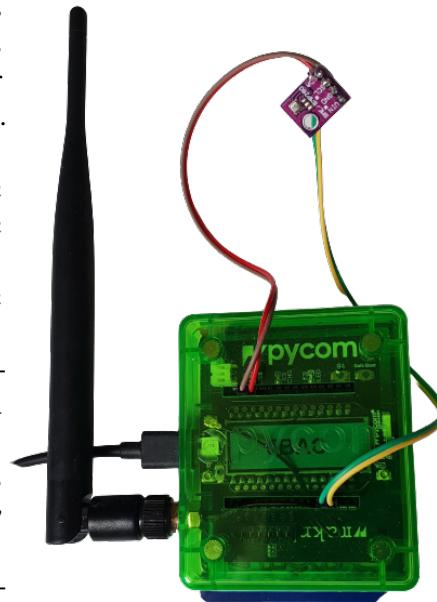
- découvrir une nouvelle catégorie de réseaux appelée LPWAN dont Sigfox et LoRaWAN sont les représentants les plus connus ;
- voir l'évolution de la pile protocolaire de l'internet qui passe de IPv4/TCP/HTTP à IPv6/UDP/CoAP tout en préservant le concept REST basé sur des ressources identifiées sans ambiguïté par des URI ;
- expliquer comment CBOR peut être utilisé pour structurer des données complexes en complément de JSON ;
- enfin JSON-LD et la base de données MongoDB nous permettront de manipuler aisément l'information collectée. Ainsi, nous introduirons les techniques essentielles pour valider statistiquement les données collectées.

À travers ce cours, vous apprendrez à programmer un Objet économe en énergie et interopérable avec d'autres Objets.

Materiel

Il n'est pas nécessaire d'avoir du matériel pour faire la plupart des exercices et expérimenter les concepts. A peu près tout peut se faire avec des scripts Python, mais ce n'est pas aussi amusant que les expérimentations en vrai. En particulier pour découvrir la magie des réseaux radios longue portée. C'est pour cela que nous allons utiliser le matériel suivant :

- un LoPy4¹ -Pycom - *Quadruple Bearer MicroPython enabled Dev Board*; Attention : prendre l'option *With headers*
- une Expansion Board 3.0² - Pycom - *Compatible with Pycom Multi-Network IoT*;
- une antenne LoRa³ (868MHz/915MHz) & Sigfox Antenna Kit - Pycom avec son petit fil permettant de la connecter au LoPy ;
- un boîtier⁴, mais ce n'est pas indispensable, si vous n'êtes pas soigneux *Pycase Clear - Fits Pycom IoT Dev Boards, Expansion Board & Antenna kit* ;
- un capteur BME280 3v3⁵ Capteur de pression température humidité BME280 - Boutique Semageek ou BME280 3.3⁶ ;
- un câble USB (USB 2.0A to micro B 2.0 1.5 m) ;
- des câbles dupont mâle-femelle⁷



Le LoPy propose un an d'abonnement gratuit au réseau Sigfox, c'est suffisant pour expérimenter. Un abonnement annuel coûte une dizaine d'euros. Par contre, accéder à un réseau LoRaWAN est plus problématique. Les offres des opérateurs ne sont pas toujours adaptées et la couvertures des réseaux communautaires n'est pas complète. Mais vous pouvez étendre cette couverture en installant votre propre antenne LoRa pour moins d'une centaine d'euros. Nous utiliserons aussi la solution proposée par Pycom.

Pour cela il faut :

- une carte d'extension pygate⁸ ;
- un LoPy comme précédemment ou un un wi-py⁹ un peu moins cher ;
- un joli boîtier¹⁰ pour faire pro ;

1. <https://pycom.io/product/lopy4/>
 2. <https://pycom.io/product/expansion-board-3-0/>
 3. <https://pycom.io/product/lora-868mhz-915mhz-sigfox-antenna-kit/>
 4. <https://pycom.io/product/pycase-clear/>
 5. <https://boutique.semageek.com/fr/704-capteur-de-pression-temperature-humidite-bme280-3009052078446.html>
 6. https://fr.aliexpress.com/item/1005002387867504.html?spm=a2g0o.productlist.0.0.580f7c3elwXr70&algo_pvid=bbd88dd7-92c5-4904-a4e7-6045b186dbd6&algo_expid=bbd88dd7-92c5-4904-a4e7-6045b186dbd6-1&btsid=0b0a182b16193744192742943e5955&ws_ab_test=searchweb0_0,searchweb201602_,searchweb201603_
 7. <https://www.amazon.fr/cable-dupont/s?k=cable+dupont>
 8. <https://pycom.io/product/pygate/>
 9. <https://pycom.io/product/wipy-3-0/>
 10. <https://pycom.io/product/pygate-case/>

- son antenne avec son câble¹¹ ;
- et cette fois-ci un câble USB-C.

0.1 Ressources disponibles

Ce livre en Open Source est accessible sur <http://source.plido.net> en version française <http://livre.plido.net> et anglaise <http://book.plido.net>. Les remarques et les commentaires peuvent être remontés en utilisant les outils de Github comme les *Issue* et les *Pull requests*.

Les vidéos référencées dans ce livre sont également disponibles sur Youtube <http://video.plido.net>. Et comme disent les Youtubers, n'oubliez pas de liker les vidéos et de vous abonner à la chaîne.

Finalement, une version plus interactive, sous forme de MOOC est accessible ici <http://mooc.plido.net>. Le MOOC permet plus d'interactivités avec des forums pour directement poser des questions et des animations pour mieux configurer le système.

0.2 Auteurs

Laurent Toutain est professeur au département Systèmes Réseaux, Cybersécurité et Droit du numérique d'IMT Atlantique, une école de l'institut Mines-Télécom. Il est membre de l'équipe OCIF (Objets Communicants - Internet du Futur) qui se focalise sur les évolutions protocolaires et architecturales de l'internet liées à la conception de nouveaux services (Smart grid, vêtements intelligents...). Après avoir travaillé sur le protocole IPv6 et les mécanismes de transition dans différents environnements, il s'intéresse actuellement à leur intégration dans l'internet des objets. Il contribue également aux Fab Labs pour l'adoption de ces protocoles. Il est l'auteur de plusieurs livres de référence sur les réseaux.



Kamal Singh est Maitre de Conférences à Télécom Saint-Étienne où il dispense des cours de réseaux informatiques et de réseaux d'opérateurs. Il fait également partie de l'équipe de recherche Data Intelligence du Laboratoire Hubert Curien. Son travail porte sur l'internet des objets, les villes intelligentes, le Big Data, le Web sémantique, la qualité de l'expérience et le software defined networking.



11. <https://pycom.io/product/lora-868mhz-915mhz-sigfox-antenna-kit/>

Marc Girod Genet est professeur associé à Télécom SudParis et chercheur associé CNRS-SAMOVAR (UMR 5157), au sein duquel il anime la thématique transverse sur l'énergie. Ses axes de recherche englobent notamment les réseaux personnels (réseaux de capteurs et architectures de mesures inclus), les communications M2M et les architectures de type IoT/WoT, les modèles de données sémantiques et les ontologies. Marc est par ailleurs impliqué dans des activités de standardisations au sein de l'AIOTI (Alliance for Internet of Things Innovation) et de l'ETSI (rapporteur, TC SmartBAN). Il a reçu en 2010 le prix spécial du jury « Croissance verte numérique » pour ses travaux sur les réseaux électriques intelligents et la gestion de la consommation d'énergie (un de ses deux domaines d'application avec la eSanté).



Patrick Maillé est professeur au département Systèmes Réseaux, Cybersécurité et Droit du numérique d'IMT Atlantique, une école de l'institut Mines-Télécom. Diplômé de l'école polytechnique et Télécom ParisTech, il a soutenu sa thèse à Télécom Bretagne (maintenant IMT Atlantique) en 2005. Ses travaux de recherche portent sur l'économie des réseaux de télécommunication à l'aide d'outils de mathématiques appliquées et d'économie (notamment la théorie des jeux).



Mireille Batton-Hubert est Professeure à l'École Nationale Supérieure des Mines de Saint Étienne, au centre d'enseignement et de recherche, l'institut Henri Fayol. Elle est responsable de l'équipe Génie Mathématique et Industriel (GMI) et rattachée à l'UMR Limos. Le département Génie Mathématique et Industriel vise à proposer des méthodes quantitatives d'évaluation de la performance globale des entreprises et de leurs produits. Il rassemble des compétences allant des mathématiques appliquées au génie industriel. Le département Génie Mathématique et Industriel vise à proposer des méthodes quantitatives d'évaluation de la performance globale des entreprises et de leurs produits. Il rassemble des compétences allant des mathématiques appliquées au génie industriel.



Vincent Lerouvillois, Assistant pédagogique





1. LES BASES DE L'INTERNET DES OBJETS (IOT)

1.1 Introduction

Dans cette première partie du cours, nous allons poser les bases de ce qu'est l'internet des objets (Internet of Things (IoT) in english). Qu'est-ce qu'on entend par IoT dans le cadre de livre ? Quelles sont les problématiques auxquelles doit répondre l'IoT et son évolution aujourd'hui ? Quelles sont les technologies, les architectures, les protocoles sous-jacents qui seront utilisés dans cet ouvrage ?

Pour cela, nous allons faire un parallèle entre la manière dont l'internet a intégré la télévision et ce que l'on vit actuellement avec l'internet des Objets.

Youtube



1.1.1 Réseaux dédiés

Dans les années 50, la télévision est devenue très populaire et presque tous les foyers ont acheté un téléviseur pour regarder leurs programmes préférés. Des réseaux de transmission dédiés ont été déployés partout sur la planète. En fait chaque type de communication avait son propre réseau un pour la radio un pour le téléphone un pour le télex,...

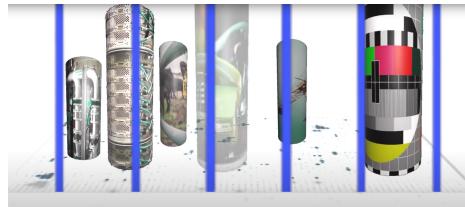
Dans les années 80, l'internet a vu le jour, mais les vitesses de transmission était faible et le réseau était limitée aux chargements de fichiers. Dans les années 90, des images et l'hypermédia avec le World Wide Web (WWW) sont apparus, en lien avec l'augmentation des débits. Toujours dans les années 90, l'augmentation en puissance des microprocesseurs a permis la numérisation du signal télé. Les téléviseurs ont commencé à inclure des microprocesseurs et les réseaux sont passés d'une transmission analogique au numérique ; mais ils sont restés dédié à cet usage unique diffuser la télévision. Avec l'entrée dans le nouveau millénaire, l'internet a gagné en débit avec l'Asymmetric Digital Subscriber



Line (ADSL) et des fibres optiques. Il était possible d'intégrer des images dans les pages web mais la qualité était médiocre. Au même moment des centaines de canaux de télévision diffusaient en haute résolution leur programme via satellite ou par Télévision Numérique Terrestre (TNT). De nos jours les communications internet ont gagné en vitesse et en qualité et certains pays ont coupé la TNT et choisi de ne transmettre leurs programmes que par internet. En fait l'utilisation d'internet n'est pas uniquement un changement de réseau de distribution c'est aussi un changement majeur dans les usages et les applications. Vous pouvez regarder la télévision sur votre téléphone portable ou même regarder votre série favorite quand vous le voulez, à la demande.

1.1.2 3 phases technologiques

De cet exemple, on peut définir trois phases dans le développement d'une technologie. Dans la première phase, un réseau spécifique est construit pour un usage bien défini. On appelle cela une approche **verticale**; une technologie est dédiée à un seul usage. Il est difficile d'échanger de l'information entre deux verticales. On fait aussi référence à des **silos** car ils sont isolés. Dans une seconde phase, les verticales commencent à intégrer des technologies communes mais pas d'une manière coordonnée. Elles ne peuvent toujours pas communiquer facilement car elles n'ont pas fait les mêmes choix.



Dans une dernière phase, des verticales se coordonnent pour converger vers les mêmes technologies en définissant des règles et des usages communs. Ceci dans le but de réduire les coûts ou d'augmenter leur impact dans ce cas. On parle d'**horizontal** car elle couvre plusieurs secteurs. L'internet est devenu une de ces horizontales pour beaucoup de services. L'internet des objets suit ce même mouvement. Des solutions particulières ont émergé pour résoudre des besoins spécifiques en agriculture, dans l'automobile, dans la santé, dans l'énergie. Quand les réseaux internet ont permis des communications à faible coût, l'architecture de l'internet a été prise en compte mais sans compatibilité. Le changement qu'on vit actuellement est la définition de fonctionnalités communes à différents domaines. Le but étant de réduire les coûts mais également de croiser les informations pour une meilleure gestion du processus industriel et un meilleur usage des ressources.



1.2 L'Internet des Objets

Comment définir l'internet des objets ? Ou plutôt, quel internet des objets allons-nous étudier ? L'ambiguïté des deux termes "internet" et "objets" impose une définition plus précise ; ou du moins une classification pour mieux comprendre à quoi l'on fait référence.

L'internet est maintenant totalement intégré dans nos vies, pour le travail, l'enseignement, pour les distractions. On l'utilise à la maison ou au travail sur nos ordinateurs, et on l'emporte avec nous de plus en plus avec nos smartphones.

Chacun a sa définition de ce qu'est internet. Pour le grand public, il peut s'agir d'applications très populaires comme Facebook, Tik-Tok, Netflix, Zoom. Pour certains, un peu plus technophiles, l'internet peut être confondu avec le Web auquel on accède via Chrome ou Firefox. Les techniciens parleront de protocoles comme IP, TCP, HTTP, et d'adresses comme les adresses IP ou les URL.

Comme ce livre est orienté technologie, notre approche relève plutôt de cette dernière catégorie. Nous verrons comment des protocoles développés il y a une vingtaine d'années pour des ordinateurs peuvent s'appliquer à d'autres dispositifs qu'il nous reste à définir.

L'objectif de l'internet des Objets est de poursuivre l'intégration du réseau Internet pour permettre à autre chose que des ordinateurs d'échanger des données. Le but principal est d'optimiser les processus pour qu'ils soient plus efficaces pour économiser des ressources ou d'augmenter la productivité. Il s'agit donc d'un internet enfoui, loin du frigo ou de la montre connectés, qui vont remonter des informations avec une infrastructure ou d'autres équipements. On peut imaginer des capteurs dans une usine pour contrôler la production, des voitures connectées qui vont dialoguer pour éviter les collisions, la mesure du taux de remplissage des bennes de recyclage dans une ville pour optimiser les circuits de collecte, la surveillance du degré d'humidité d'un champ pour réduire la consommation d'eau...

L'internet des Objets peut se résumer de la manière suivante : utiliser des protocoles développés pour des ordinateurs et maintenant des téléphones portables (plus puissants que les ordinateurs utilisés par l'internet à ses débuts) mais dans des environnements plus contraints. En effet, les lois de Moore, définissant les puissances de traitement des processeurs ainsi que la diminution continue des coûts de la mémoire, nous ont permis de doter les petits objets de ressources comparables à celles des ordinateurs d'il y a trente ans.

L'internet des Objets, c'est résoudre l'équation suivante : continuer à faire la même chose car tous les systèmes d'information actuels utilisent les mêmes principes mais le faire différemment car ces principes sont trop coûteux en énergie, en temps de calcul et échange de données.

L'IoT, l'internet des objets ou des choses, est une architecture globale permettant à des objets (équipements de même type ou non), d'interagir de manière autonome via internet. Cette interaction :

- est réalisée, par construction, au travers d'un réseau internet, ce qui implique généralement que les objets/choses soient pourvus d'une adresse IP ;
- est relatif à des commandes (opérations de contrôles ou appels de fonctions) ou des échanges de données ou d'informations.

Ce nouveau paradigme Sciences et Technologies de l'Information et de la Communication (STIC) qu'est l'IoT est une convergence de nombreux domaines d'applications tels que : les maisons ou bâtiments intelligents, les villes du futur, l'industrie du futur (industrie 4.0), l'énergie, les systèmes de transport, l'agriculture, la eSanté, etc., vers une suite protocolaire réduite, interopérable et sécurisée. Le mouvement est en marche et, vu le nombre d'acteurs concernés, va prendre plusieurs années. Mais les bases sont déjà bien établies et c'est ce que vous allez apprendre dans cet ouvrage.

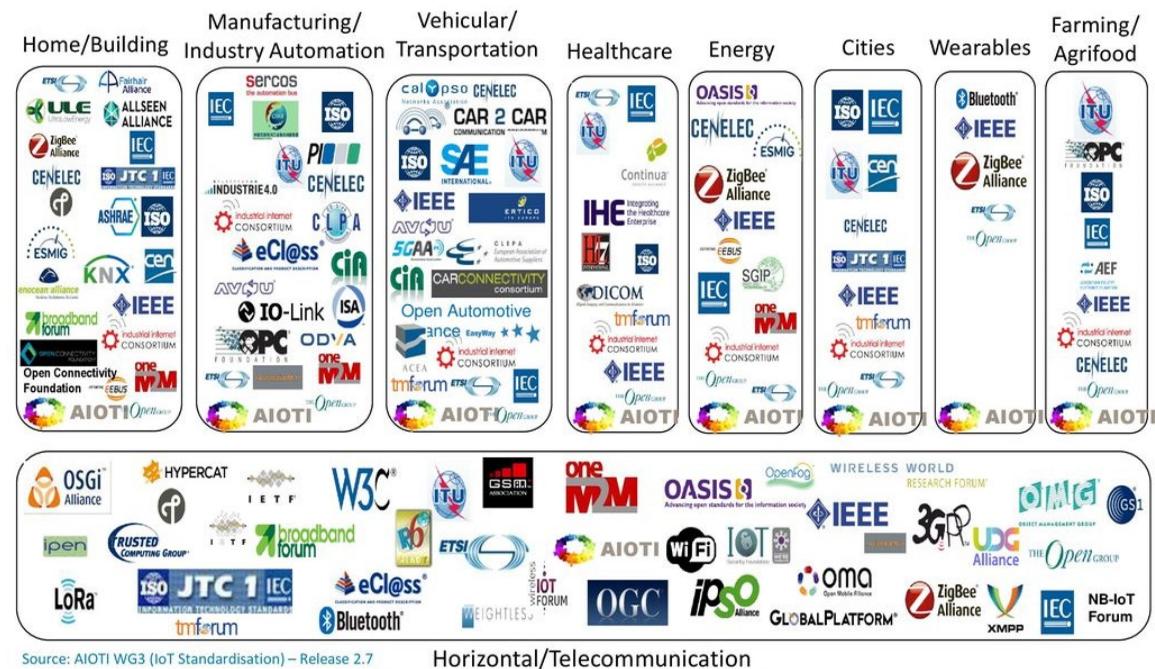


FIGURE 1.1 – Quelques standards de l’IoT

1.3 Le problème

Un des problèmes que rencontre l’internet des objets, c’est que l’IoT ne démarre pas ex-nihilo. Maintenant que les technologies qui ont fait le succès de l’Internet sont matures, il ne s’agit pas juste de les appliquer à un nouveau domaine. Des objets étaient capables de communiquer bien avant qu’internet n’existe. Chaque secteur a déjà développé ses solutions, plus ou moins standards, plus ou moins propriétaires.

La figure 1.1 reprend un certain nombre de travaux et de groupes qui spécifient les protocoles pour l’internet des objets.

Sans entrer dans les détails, on voit que certains logos se retrouvent à plusieurs emplacements, qu’il y a pour chaque secteur une profusion de solutions qui nuisent à l’interopérabilité et aux évolutions. L’internet des objets, dans son acceptation la plus large, consiste à simplifier cette architecture, comme l’internet l’a fait il y a quelques années dans le domaine des télécoms en simplifiant ce modèle et en permettant à ces différents acteurs de converger vers une architecture commune et un ensemble de solutions plus réduit.

Cela ne veut pas nécessairement dire moins d’acteurs, mais une plus grande cohérence dans les choix technologiques.

La figure 1.2 page suivante analyse l’IoT par domaine d’application en se focalisant sur la composante réseau. L’IoT et les objets connectés sont des systèmes complexes pour lesquels les solutions open source, les alliances entre industriels, les organismes de standardisation sont toujours fragmentés ; cependant de manière moins importante, montrant que la structuration est en marche.

Cette fragmentation de l’écosystème est paradoxale. Si l’on résume sommairement, les solutions proposées reviennent à interroger un équipement sur le terrain pour accéder à une valeur, la traiter et renvoyer une commande pour interagir avec l’environnement.

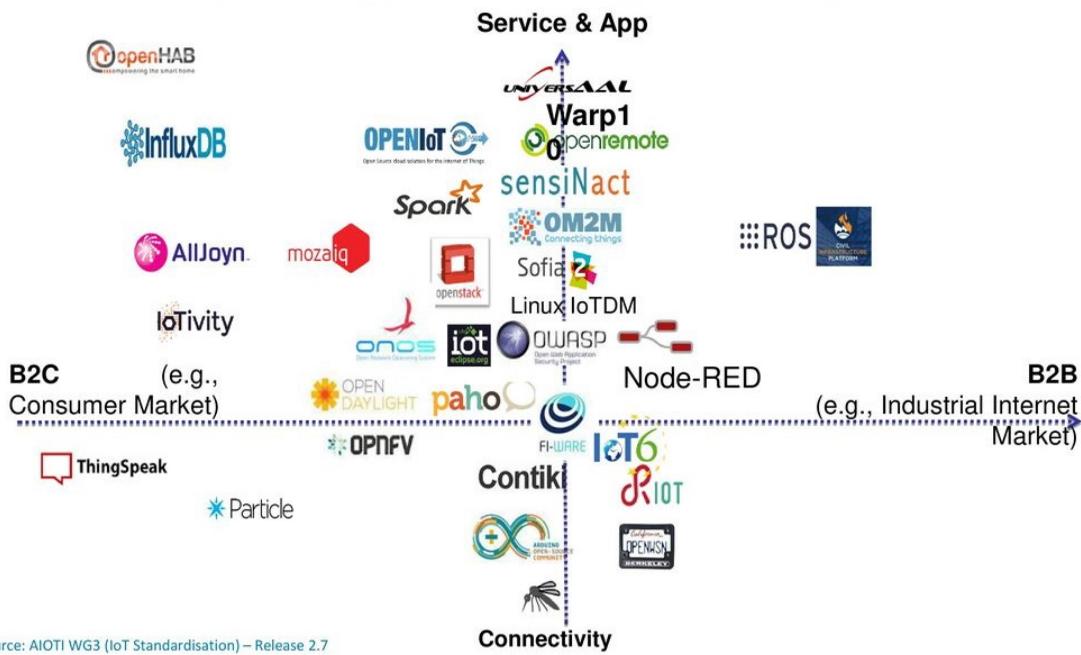


FIGURE 1.2 – Quelques applications Open Source pour l'IoT

Pourquoi cette foison de solutions différentes ? Cela peut venir des besoins de fiabilité, de sécurité, de portée de la donnée, mais cela vient aussi de l'histoire. La communication avec des Objets est tout aussi ancienne que la communication entre ordinateurs (qui sont eux-mêmes des Objets). Mais à l'époque, chaque domaine a suivi sa propre voie en spécialisant les solutions pour répondre à ses besoins propres. Il en résulte des solutions optimisées pour un domaine particulier. Mais chaque fois qu'il faut modifier une technologie, le travail doit être adapté pour chaque domaine, introduisant des coûts et des délais supplémentaires.

De même, chaque domaine ayant sa propre représentation des données, il est relativement difficile de les combiner pour avoir une vision plus globale. On arrive donc à des systèmes fermés, chers, peu évolutifs, mais optimisés pour les tâches qu'ils ont à réaliser.

Au fil du temps, les protocoles de l'internet ont pu être utilisés, mais il s'agit surtout de complémer les technologies existantes sans qu'il soit possible d'interconnecter deux domaines.

1.4 Evolution de l'IoT

L'exemple de l'évolution du réseau de télévision est éloquent. À ses débuts, ce réseau est analogique et hautement spécialisé pour diffuser les programmes transportés sur des signaux analogiques sur des équipements spécialisés, les téléviseurs.

Avec les progrès des processeurs informatiques, il devient possible de transporter les données en utilisant un codage numérique. Mais des réseaux spécialisés restent nécessaires, l'Internet n'offrant pas la même qualité.

La dernière étape consiste à intégrer ces flux dans l'internet classique. Cela devient possible par

la montée en débit des réseaux filaires et radio (Wi-Fi, 4G...).

Cette mutualisation des accès via l'internet permet non seulement une réduction des coûts, mais aussi l'apparition de nouveaux usages comme la télévision sur les téléphones portables ou les séries à la demande.

L'internet des Objets suit la même voie. En plus des technologies spécifiques, les protocoles de l'internet sont intégrés, mais en les adaptant aux contextes du secteur. Nous sommes actuellement en train de vivre la convergence vers un ensemble réduit de protocoles, une standardisation de la représentation de la donnée, et son traitement sur des plates-formes plus génériques.

Le déclencheur n'est pas la montée en débit comme pour la télévision, mais la possibilité d'avoir des équipements peu chers, aux capacités réduites par rapport à l'informatique traditionnelle et autonomes énergétiquement, tout en ayant une meilleure intégration dans les systèmes d'information actuels.

1.5 Des objets contraints

Avec les progrès de l'électronique, les processeurs deviennent de plus en plus puissants et les super-ordinateurs d'hier sont maintenant dans une montre ou un smart-phone.

Pour l'internet des objets, la logique est un peu différente. La loi de Moore va induire une réduction des coûts de fabrication plutôt qu'une augmentation des puissances de traitement. Le principal critère pour un internet des objets massif reste l'énergie ; connecter un appareil à une source d'énergie ou recharger une batterie a un coût. Augmenter la vitesse du processeur ou la taille de la mémoire induit une plus grande consommation d'énergie de l'objet. On peut donc s'attendre à une certaine stabilité des performances des objets car ceux-ci resteront limités en performances.

Les objets sont généralement limités en termes de puissance de traitement, de mémoire et d'énergie. Selon le standard de l'IETF [RFC 7228](#), les dispositifs peuvent être répartis en trois classes qui se retrouvent aussi dans la segmentation des processeurs :

- La classe 0, avec moins de 10 ko de mémoire volatile pour stocker les données temporaires et 100 ko de mémoire Flash pour stocker le code informatique de l'objet. C'est l'équivalent d'un **Arduino** UNO (2 ko de RAM et 32 ko de Flash). Il est presque impossible d'installer à la fois les protocoles utilisés pour communiquer sur Internet (même de manière restreinte) et les applications qui tournent dessus.
- La classe 1 a environ 10 Ko de RAM et 100 Ko de Flash. Avec une adaptation, il est possible d'y installer une pile IP. Il s'agit par exemple d'équipement comme le Pycom Lopy4 que nous utiliserons par la suite (et qui se situe dans la limite haute) sur lequel le système d'exploitation est minimal. Ainsi, le Pycom utilise une version simplifiée du langage Python (micro-python) qui permet de l'adapter à la limitation du système.
- La classe 2 est moins restreinte avec au moins 50 ko de RAM et 250 ko de Flash (comme un **Raspberry Pi**). Le système d'exploitation **Linux** peut fonctionner sur ces appareils. Par conséquent, il y a peu de limitations sur la pile IP et les applications s'y exécutant. Les appareils de classe 1 ont trop de restrictions pour utiliser les protocoles définis pour des objets plus gros. L'Internet Engineering Task Force (IETF), l'organisme qui standardise les protocoles de l'internet, a proposé une révision de sa pile de protocoles afin d'adapter sa pile protocolaire à un environnement contraint.

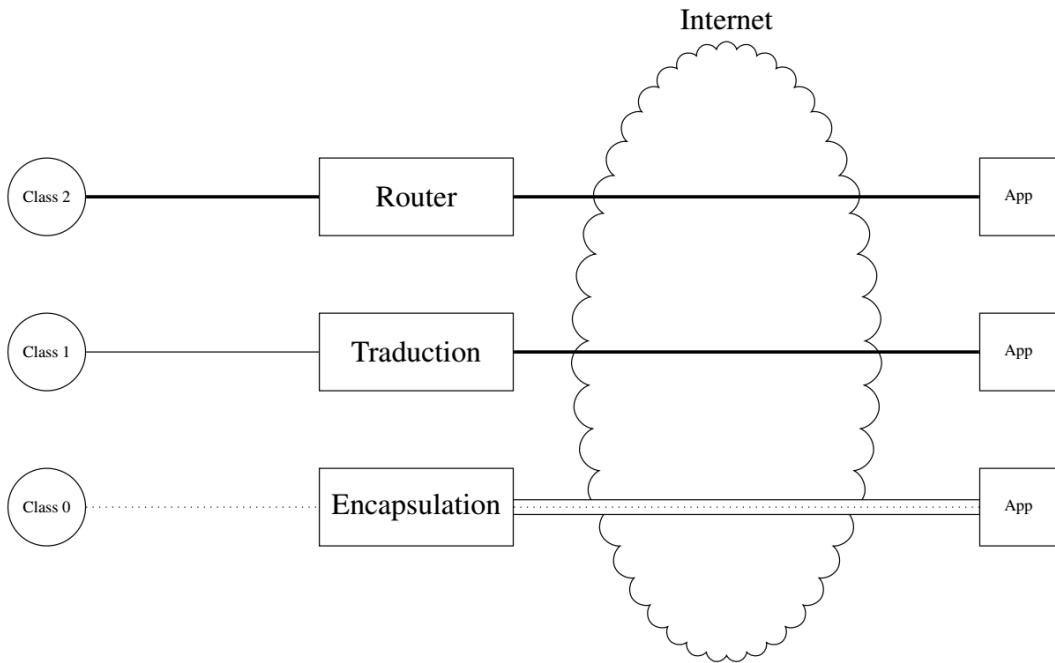


FIGURE 1.3 – Possibilités d’interconnection

La figure 1.3 résume les moyens d’interconnexion suivant la classe de l’objet :

- Un appareil de classe 0 ne peut pas utiliser directement l’internet pour échanger des informations, d’où la nécessité d’installer une passerelle pour capter le trafic et l’envoyer sur l’internet. Il ne possède pas directement d’adresse IP. Les passerelles LoRaWAN LoRaWAN Network Server (LNS) et 3GPP Service Capability Exposure Function (SCEF) agissent dans ce sens (nous reviendrons là-dessus). Les données produites sont encapsulées par ces passerelles dans des protocoles comme HyperText Transport Protocol (HTTP) ou Message Queuing Telemetry Transport (MQTT), que nous verrons également dans la suite du cours.
- Les appareils de classe 1 peuvent aussi utiliser une passerelle pour s’interconnecter à l’internet traditionnel, mais plutôt que d’encapsuler les données produites dans d’autres protocoles comme le fait la classe 0, les passerelles pour les appareils de classe 1 vont traduire un protocole contraint dans son équivalent dans le monde non contraint. Les dispositifs de classe 2 peuvent interagir directement avec d’autres nœuds sur l’internet, sans passer par une passerelle.

1.6 L’interopérabilité

Un autre défi concerne le nombre de dispositifs. Certaines études prévoient 500 milliards de dispositifs à la fin de la décennie.

INTEROPERABILITY

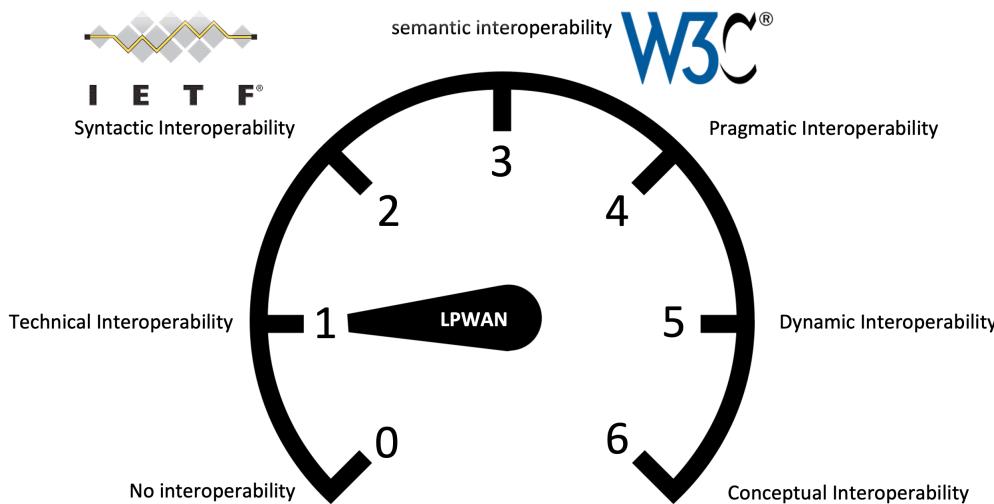


FIGURE 1.4 – Niveau d'interopérabilité

Avec cet énorme internet des objets, où presque chaque équipement comprendra des éléments de détection ou d'action, l'intégration dans un système d'information deviendra un véritable défi. Comme l'internet des objets actuel est conçu pour une verticale (c'est-à-dire pour des applications spécifiques), les dispositifs sont choisis et intégrés au moment de la conception du système. Les ingénieurs choisissent leurs capteurs, connaissent précisément leurs caractéristiques et écrivent leur code en fonction de ce qu'ils ont intégré.

L'internet des objets massif change la donne. Les dispositifs ou les choses ne peuvent pas être intégrés dès le départ dans un système d'information statique. L'intégration doit se faire au fil du temps et doit gérer les évolutions des dispositifs pendant longtemps (les fabricants de dispositifs peuvent changer, les produits évolueront avec de nouvelles fonctionnalités, etc.).

L'internet qu'on connaît est une très bonne illustration du besoin d'interopérabilité. Il a permis, grâce à une uniformisation du réseau et une force baisse des coûts de transmission, de développer de nouveaux usages. Vous n'auriez jamais investi pour un réseau propre à la vidéoconférence et au télétravail. Mais en mutualisant les usages, c'est possible ! L'internet des objets doit suivre la même voie et également converger vers l'architecture de l'internet actuel. Le mot clé est donc l'interopérabilité.

Cette question d'interopérabilité a été formalisée dans le modèle Levels of Conceptual Interoperability Model (LCIM) [TM03] (voir figure 1.4) peut être représenté par un compteur pour mesurer le degré d'interopérabilité.

Il distingue six niveaux d'interopérabilité, parmi lesquels :

- Au niveau zéro, on n'est pas connecté ; on ne parle à personne donc on n'a pas de problèmes d'interopérabilité.

Youtube



- Au niveau 1, on est capable de transmettre de l'information, mais il faut que les deux côtés connaissent les règles. On a un système intégré ; les applications doivent connaître précisément les spécifications des objets avec lesquels ils communiquent, car elles définissent ses propres formats des échanges de données. On pourrait prendre l'exemple qu'une carte électrique où un processeur communique avec des capteurs via un circuit imprimé. Le code tournant sur le processeur peut être écrit à l'avance car il y a peu de chance qu'un utilisateur dessoude les composants pour les remplacer par d'autres. Cela correspond à l'encapsulation, si l'on considère les objets en réseau de la figure 1.3 page 22, l'élément d'interconnexion doit être configuré pour en fonction de l'objet émettre encapsuler les données vers la bonne application chez le bon récepteur.
- L'interopérabilité syntaxique (niveau 2) où deux nœuds peuvent échanger des données, sans être préalable configurés pour cet échange. C'est le cas de l'Internet. En utilisant cette suite de protocoles, en ayant une adresse valide sur le réseau, toute application est capable d'échanger des données avec une autre. En revanche, les données que vous allez échanger sont propres à une application. Les vidéoconférences sont un très bon exemple d'interopérabilité de niveau 2. Vous ne pouvez pas utiliser Zoom si votre correspondant utilise Teams car les formats sont différents. L'IETF est le regroupement de différents acteurs (industriels, académiques,...) qui produisent les standards liés à ce réseau. Il se reconnaissent par l'acronyme Request For Comments (RFC) suivi d'un nombre.
- L'interopérabilité sémantique (niveau 3) implique que le récepteur est capable d'interpréter les données reçues. Le web est un très bon exemple d'interopérabilité de niveau 3. Quel que soit votre navigateur, vous pouvez afficher les pages d'un site web et suivre les liens. Le sens de l'information est compris de la même manière des deux côtés. Pour le Web, le format HyperText Markup Language (HTML) permet à l'aide de balises (mots-clés) de structurer un texte en y ajoutant des informations de formatage ou des liens vers d'autres documents. Le World Wide Web Consortium (W3C) définit les standards.
- Les niveaux supérieurs d'interopérabilité vont être liés à la précision du modèle qui va représenter le système.

1.7 Le besoin de standardisation

Les objets n'ont pas attendu l'internet pour communiquer. Ils ont évolué chacun dans leur verticale, développant des solutions satisfaisantes mais limitées en termes d'évolution et d'interopérabilité. Cet état des lieux sur la dispersion des écosystèmes met en avant la nécessité :

- d'une coordination entre les alliances ;
- d'une harmonisation ou d'un alignement des standards ;
- de disposer au plus vite d'implémentations de référence et de modèles de référence.

Dans le cas contraire, l'interopérabilité ne pourra pas être traitée comme il se doit, les nouveaux services innovants multi-domaines ne seront pas couverts, et le développement de l'IoT pourrait être freiné voire avorté.

Les aspects liés à l'éducation et à la formation des acteurs de l'IoT, ainsi que ceux liés à l'acceptation, aux usages et évidemment au volet socio-économique de l'IoT, sont aussi des points essentiels dont il faut tenir compte.

Les organismes de normalisation tels que l'IETF ou le W3C ont conçu des protocoles ou des modèles de données capables de traiter l'interopérabilité. D'une certaine manière, c'est la clé du succès de l'internet actuel. Il a permis de résoudre le problème de l'interopérabilité au niveau syntaxique et sémantique mais au prix de messages volumineux.

Le défi pour l'internet des objets est de s'intégrer dans ce système distribué géant. Comme l'internet des objets est un nouveau venu, l'évolution devra se faire de son côté pour prendre en compte les règles existantes, mais en les adaptant. Les prochains chapitres traiteront de ces changements.

1.8 Questions

Question 1.8.1: Nouveau Domaine

Les objets communicants sont un tout nouveau domaine, lié aux progrès en miniaturisation des composants électroniques :

- Vrai
- Faux

Question 1.8.2: Protocoles

Laquelle de ces affirmations est vraie ?

- Il y a très peu de protocoles pour faire communiquer les Objets. Comme l'Internet est une technologie qui s'est très répondu, son succès va permettre aux objets de communiquer.
- Il y a beaucoup de solutions pour permettre à des objets de communiquer, l'Internet des Objets doit permettre de les fédérer.

Question 1.8.3: Source de données

Quelle est la principale source de création des données dans l'IoT ?

- les capteurs
- les nano-ordinateurs (type Raspberry Pi)
- Internet
- les serveurs Web

Question 1.8.4: Défis

Quels sont les principaux défis technologiques pour l'Internet des Objets (3 réponses) ?

- Avoir une consommation d'énergie faible.
- Avoir une architecture protocolaire simplifiée.
- Pouvoir fonctionner sur des systèmes d'exploitation ouverts comme Linux.
- Permettre de sécuriser les données qui peuvent être sensibles.
- Transmettre en permanence leur état et des valeurs mesurées.

2. ARCHITECTURE DE L'INTERNET

2.1 Les protocoles

Vous connaissez sûrement le principe d'empilement protocolaire dans les réseaux. Chaque protocole fournit un service et se base sur celui de la couche inférieure pour le réaliser. Le modèle d'origine définit sept couches pour transporter les données d'une application, n'importe où dans le monde. Les protocoles réseaux sont empilés les uns sur les autres, ceux du dessus utilisent les services offerts par ceux d'en dessous pour acheminer la donnée. Cela a donné lieu au modèle de référence de l'International Standardization Organization (ISO) qui structure les réseaux depuis les années 1970. En théorie, il y a 7 **couches**, mais l'internet a fait évoluer ce modèle et les numéros des couches, associés à des fonctionnalités, sont restés ; ce qui peut conduire à une numérotation étrange.

L'internet a simplifié cette architecture (cf. figure 2.2 page suivante). C'est pour ça que l'on retrouve moins de couches et que les numéros ne sont pas contigus.

Les deux premières couches en partant du bas, regroupées sous le nom d'Interface, permettent de transmettre les données binaires sur un support physique. La couche 1 s'occupe de cette modulation sur un support physique

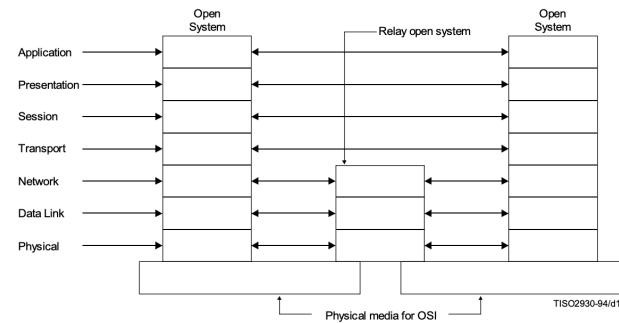


Figure 12 – Communication involving relay open systems

FIGURE 2.1 – Extrait du standard ITU-T Rec. X.200 (1994 E)

Youtube



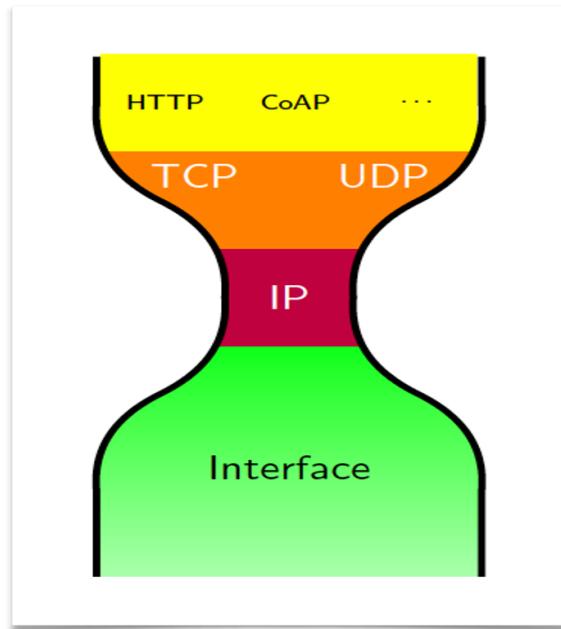


FIGURE 2.2 – Empilement protocolaire de l’Internet

particulier (fibre optique, paire de cuivre, onde radio). La couche 2 regroupe les mécanismes qui permettent de structurer cette donnée sous forme de bloc de taille finie appelées trames, de définir les méthodes d'accès, c'est-à-dire quand l'équipement peut émettre, et les formats des adresses utilisées pour identifier les équipements.

- l’Institute of Electrical and Electronics Engineers (IEEE) qui propose des standards comme Ethernet pour les réseaux filaires ou Bluetooth et Wifi pour les réseaux radio,
- le 3rd Generation Partnership Project (3GPP) qui opère au même niveau et définit les protocoles pour la téléphonie cellulaire (4G),
- ...

Au-dessus, on a le protocole Internet Protocol (IP) standardisé par l’IETF. Le protocole IP s’adapte simplement à tout moyen de communication. IP propose ainsi une abstraction des moyens de communication aux couches applicatives, rendant l'accès au réseau et l'adressage universels. Le traitement dans les **routeurs** (équipements chargés d'aiguiller l'information dans le réseau) doit être le plus rapide possible pour traiter un maximum de paquets par seconde. De plus, IP ne spécialise pas le réseau pour un service ou un autre ; il ne fait qu'aiguiller les paquets vers la bonne destination. Le réseau Internet est un réseau mondial construit autour de ce protocole permettant potentiellement d'atteindre tous les équipements qui y sont connectés.

Les experts de l'internet aiment cette représentation en **sablier** où IP apparaît en position centrale mais est plus petit comparé aux autres protocoles. Par conception, IP est très simple ; à la fois pour être portés facilement sur de nombreux niveaux 2 et être facilement utilisable par les couches hautes, mais également pour traiter les données très rapidement dans les nœuds d'interconnexion.

IP est mis en oeuvre partout sur Internet aussi bien dans les équipements en extrémité du réseau que dans les routeurs chargés d'envoyer les données vers la bonne destination.

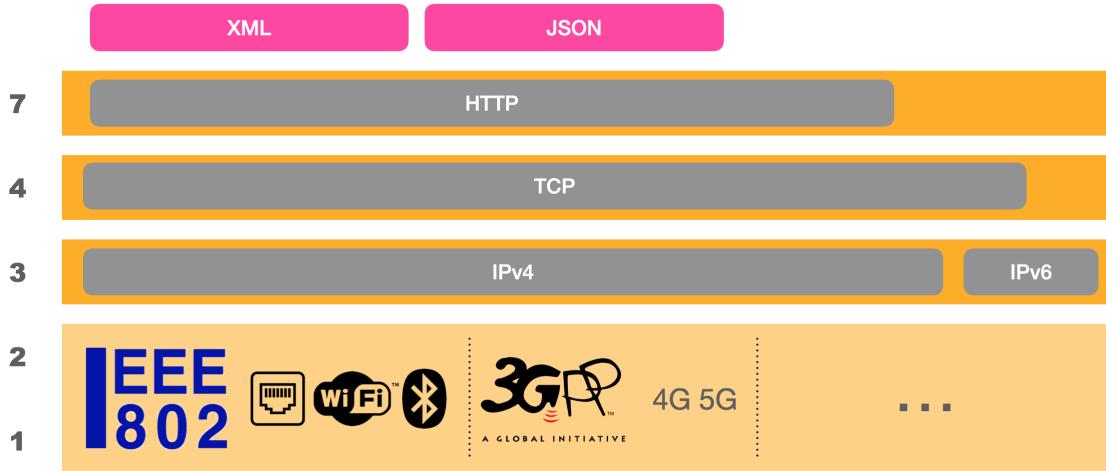


FIGURE 2.3 – Principaux protocoles de l'Internet

Au-dessus on trouve deux protocoles qui ne sont mis en oeuvre que dans les équipements d'extrémité. Si le niveau 3 permet de joindre une machine, le niveau 4 va permettre d'identifier l'application qui doit traiter les données. Les "adresses" de ces applications sont des numéros compris entre 1 et 65535 appelés **ports**. Par exemple, les serveurs Web utilisent le port numéro 80 ou le numéro 443.

Le protocole Transmission Control Protocol (TCP) va surveiller les données transférées et sera capable de retransmettre des données perdues, ralentir ou accélérer le transfert de données s'il détecte une saturation du réseau. En revanche, sa mise en œuvre est complexe et coûteuse en mémoire. Dans les cas simples, User Datagram Protocol (UDP) est préféré ; il n'apporte pas de traitement supplémentaire UDP, c'est un protocole minimal qui se contente d'aiguiller les données vers la bonne application sans aucun autre contrôle.

Au-dessus, on trouve les applications qu'historiquement on classe dans la couche 7. Les applications sont très nombreuses mais la plus répandue est HyperText Transport Protocol (HTTP) qui sert à transporter les pages web, mais également elle permet des communications directes entre ordinateurs.

Pour le grand public, l'internet désigne surtout la totalité de cet assemblage protocolaire et est souvent confondu avec l'application qui a démocratisé son usage : le Web. C'est vrai également pour les techniciens, le trafic produit par le Web est largement présent dans l'Internet. Ce schéma, figure 2.3 reprend la pile protocolaire majoritairement utilisé dans l'internet. On voit qu'au niveau 3 on a deux versions du protocole IP ; la version 4 est la version historiquement déployée et elle a eu tellement de succès qui est de plus en plus difficile d'avoir des adresses Internet Protocol version 4 (IPv4) pour les machines. Pour permettre au réseau de continuer de fonctionner, une nouvelle version a été développée. Internet Protocol version 6 (IPv6) rend l'adressage quasi infini avec des adresses sur 128 bits. IPv6 gagne petit à petit du terrain dans les usages classiques et c'est surtout

une brique essentielle pour l'internet des objets.

Le Web utilise majoritairement le protocole HTTP. Et comme HTTP repose sur TCP, ces deux protocoles sont dominants sur le réseau.

Finalement ce graphique ajoute une couche supplémentaire, au-dessus de la couche 7, pour indiquer comment les données transportées sont structurées avec des formats comme Extensible Markup Language (XML) ou JavaScript Object Notation (JSON) que nous verrons dans la suite.

Question 2.1.1: Pile Protocolaire

Dans la pile protocolaire de l'internet, quels protocoles ont pour fonction d'aiguiller les paquets jusqu'à leur destination (2 réponses)

- | | | | |
|-----------------------------------|-------------------------------|-------------------------------|-------------------------------|
| <input type="checkbox"/> Ethernet | <input type="checkbox"/> IPv4 | <input type="checkbox"/> MQTT | <input type="checkbox"/> JSON |
| <input type="checkbox"/> IEEE | <input type="checkbox"/> IPv6 | <input type="checkbox"/> HTTP | |
| <input type="checkbox"/> 802.15.5 | <input type="checkbox"/> UDP | <input type="checkbox"/> CoAP | |
| <input type="checkbox"/> Wi-Fi | <input type="checkbox"/> TCP | <input type="checkbox"/> XML | |

2.2 Les fondements du Web

L'architecture qui a conduit au Web est une formidable source d'inspiration pour le développement de nouveaux services car il est l'un des plus grands succès reposant sur le réseau Internet. Le Web forme de grands systèmes distribués et repose sur plusieurs principes qui le rendent universel et évolutif. La navigation avec un navigateur n'est que la partie visible du trafic ; les principes du web sont également utilisés pour le streaming vidéo, les échanges entre ordinateurs.

Le Web et ses extensions sont basés sur un modèle client-serveur. Les serveurs possèdent des ressources et les clients peuvent y accéder ou les modifier grâce à un protocole tel que HTTP. Le modèle client-serveur est quelque chose de courant dans les réseaux informatiques, mais le Web suit certaines directives de conception connues sous le nom de REpresentational State Transfer (REST).

Selon Roy Fielding, qui a défini ce modèle [Fie00], REST est un ensemble de principes, de propriétés et de contraintes. REST utilise le modèle de communication client-serveur et utilise généralement le protocole HTTP.

Le principe REST permet de concevoir des serveurs évolutifs. Un serveur doit être sans état, ce qui signifie qu'il ne conserve pas d'information après avoir répondu à une demande d'un client. Cela permet de simplifier le traitement dans le serveur qui doit traiter les requêtes d'un grand nombre de clients.

Cela impose que l'état soit situé du côté du client. Cet état est alimenté à partir des données structurées que le client reçoit du serveur. Ainsi, lorsqu'un client demande une page Web, celle-ci peut contenir d'autres Universal Resource Identifier (URI) pour la compléter, par exemple des images, des feuilles de style, des scripts, etc.

Le client doit donc comprendre les données que le serveur lui envoie et donc connaître le format de représentation de la ressource qu'il reçoit pour y retrouver les URI. Donc, en plus de la ressource elle-même, le serveur ajoute des informations complémentaires, appelées métadonnées.

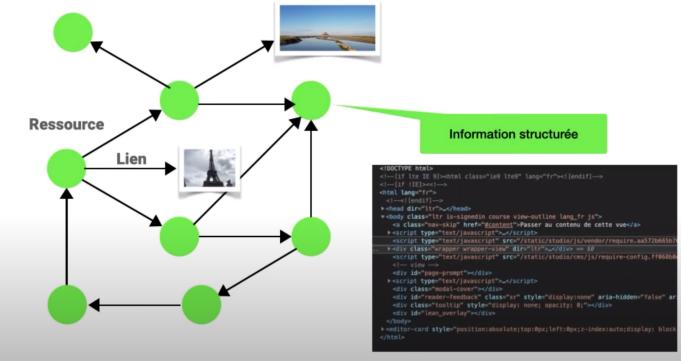
Youtube



Elles intègrent entre autres le format du contenu (content format). Il peut s'agir de texte pur, d'une image ou d'un format de texte structuré tel que HyperText Markup Language (HTML) ou JSON.

2.2.1 Ressources

L'élément de base est la ressource que l'on peut définir comme un bloc de données de taille finie. Les ressources peuvent elles-mêmes contenir des références à d'autres ressources qui à leur tour vont faire référence à d'autres ressources etc. Cela forme un maillage entre ressources qui est comparé à une toile d'araignée (web en anglais). La ressource peut être, par exemple, une image dans ce cas elle ne fera pas référence à autre chose. Pour faire référence à une autre ressource, son contenu doit être structuré et doit donc être défini dans un format où l'on peut facilement comprendre qu'une partie du contenu est une référence à une autre ressource. HTML est un de ces langages qui permet aux pages web de se référencer entre elles par le biais de liens.



2.2.2 Identifiants

Chaque ressource du Web est identifiée par une valeur unique appelée URI. Si l'URI contient des caractères internationaux, (comme les lettres accentuées, ...) il est appelé International Resource Identifier (IRI).

Les URI permettent de désigner une ressource de manière non ambiguë, c'est-à-dire que l'on ne retrouvera pas le même URI pour désigner deux ressources différentes. Par construction, la structure de l'URI est hiérarchique, ce qui permet de créer des identificateurs uniques de manière distribuée. Si vous voulez identifier une ressource, vous devez posséder une séquence unique : un numéro de téléphone, un numéro de sécurité sociale, un nom de domaine. En y ajoutant quelque chose d'unique pour nous, cela crée un identifiant globalement unique. The URI allow to designate a resource in an unambiguous way, i.e. one will not find the same URI to designate two different resources. By construction, the structure of the URI is hierarchical, which allows unique identifiers to be created in a distributed fashion. If you want to identify a resource, you must have a unique sequence : a phone number, a social security number, a domain name. By adding something unique to us, it creates a globally unique identifier.

Par exemple, pour identifier une image, on peut la nommer

image

mais il y a peu de chance que ce nom soit unique, d'autres personnes sur Terre ont sûrement eu la même idée. En revanche, si je la fais précéder de mon numéro de téléphone

33667789078 image



FIGURE 2.4 – Structuration d'une URI

sera unique si je ne nomme qu'une seule ressource "image". Un autre utilisateur sur le même principe pourra nommer sa ressource :

33667239018 image

sans ambiguïté possible. Cependant, comme le numéro de téléphone est unique dans l'espace des numéros de téléphone, d'autres numéros uniques pourraient entrer en conflit dans d'autres espaces de numérotation.

Pour éviter les conflits, il est intéressant de donner, au début l'espace de numérotation, par exemple :

tel : 33667789078 image

et

ss : 33667789078 image

les deux identifiants seront uniques, même si le hasard a fait que ce numéro de téléphone et ce numéro de sécurité sociale coïncident.

Les URI formalisent ce principe. Le [RFC 3986](#) explique comment ils peuvent être construits. Un URI commence par un schéma indiquant l'autorité de nommage, suivi d'une valeur d'autorité puis d'un chemin dans l'espace d'autorité. Des caractères comme les ":" ou les "/" sont utilisés pour améliorer la lisibilité de l'URI.

Par exemple :

mailto : mduerstifi.unizh.chssh://utilisateurexample.com
ftp://ftp.is.co.za/rfc/rfc1808.txt

Ainsi, si je mets une ressource sur un site Web, celui-ci est identifié par un nom de domaine, par exemple example.com. Je suis propriétaire de ce nom. Je peux donc l'utiliser pour identifier de manière unique ma ressource. Si on reprend le principe de construction d'un URI, j'aurai : So, if I put a resource on a Web site, it is identified by a domain name, for example example.com. I own this name. I can therefore use it to uniquely identify my resource. If we take the principle of construction of a URI, I will have :

http://example.com/ma_ressource

Personne d'autre dans l'univers ne pourra identifier ses ressources avec cette chaîne de caractères puisque example.com m'appartient. Je dispose donc d'un espace de nommage infini qui me permet de désigner l'ensemble infini de ressources sans que personne d'autre ne puisse prendre les mêmes noms. Un URI est une construction administrative permettant d'attribuer un identifiant unique global à une ressource spécifique.

L'URI (cf.figure 2.4 page précédente) a pour but de facilement nommer une ressource, de pouvoir lier les ressources entre elles pour former cette toile d'araignée mondiale. Le schéma définit à la fois l'espace de nommage de l'autorité et son format. Une adresse IP ou un nom de domaine comme autorité est à la fois un moyen d'assurer l'unicité globale, mais également de savoir comment accéder à la ressource.

Par exemple, **spotify** a défini son propre schéma et ensuite il n'a plus besoin d'autorité mais structure le chemin pour référencer une playlist.

Tous les livres ont un numéro International Standard Book Number (ISBN) qui permet d'identifier. Il peut être intégré aussi dans une URI. Il faut voir que ces deux types d'identifiants permettent de référencer un objet unique mais rien qu'en le lisant on ne peut pas accéder à la ressource. On appelle cette sous famille des URI, des Univeral Resource Name (URN).

Un sous-ensemble d'URI peut être directement utilisé pour localiser la ressource, c'est-à-dire trouver sur quel serveur se trouve la ressource et comment y accéder. Il s'agit d'une Univeral Resource Locator (URL) bien connue du grand public et utilisée par les navigateurs Web.

Le schéma http est bien pratique car il peut se lire également comme un URL. Ce schéma donne :

- le protocole à utiliser pour accéder à la ressource (http),
- l'autorité qui indique l'adresse du serveur (et son port),
- et enfin, le chemin d'accès de ce que l'on va demander au serveur et qui peut parfois correspondre à une arborescence de fichiers sur un serveur.

Mais il faut bien voir que le but initial est de faire un identifiant unique. Le schéma https donne la manière dont sera construit la suite et dans un second temps uniquement sera vu comme le protocole à utiliser pour accéder à la ressource. L'autorité est unique et dans un second temps servira à localiser le serveur. Et finalement le chemin va indiquer comment parvenir à accéder à la ressource sur le serveur. Donc les ressources de notre toile d'araignée mondiale sont présentes sur des serveurs et chaque ressource possède un identifiant unique. Dans un premier temps, le client connaît l'URI d'une ressource. Si c'est une URL, il peut contacter le serveur. Le serveur lui retourne la ressource. Le client l'analyse et découvre les URI qu'il contient. Il peut donc interroger le ou les autres serveurs pour reconstruire localement une partie de la toile nécessaires au traitement que le client veut effectuer.

```
▼ Hypertext Transfer Protocol
  ▶ GET /site/kamalsingh25/ HTTP/1.1\r\n
    Host: sites.google.com\r\n
    User-Agent: Mozilla/5.0 (X11; Linux i686; rv:45.0) Gecko/20100101 Firefox/45.0\r\n
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
    Accept-Language: en-US,en;q=0.5\r\n
    Accept-Encoding: gzip, deflate\r\n
    Connection: keep-alive\r\n
  \r\n
  [Full request URI: http://sites.google.com/site/kamalsingh25/1
  [HTTP request 1/1]
```

FIGURE 2.5 – Contenu d'une requête HTTP GET

Question 2.2.1: Unicité

Qu'est ce qui est unique dans le monde (6 réponses) ?

- Un prénom.
- Un nom de famille.
- un numéro de sécurité sociale utilisé en France.
- un numéro de passeport.
- un numéro de téléphone portable avec son préfixe international.
- un numéro complet de compte en banque (International Bank Account Number (IBAN)).
- l'adresse IP de ma machine dans mon réseau privé.
- l'adresse IP d'un serveur Coursera (13.225.34.28).
- le nom de domaine plido.net.
- le nom d'une ville.

2.2.3 Interactions

Les interactions entre clients et serveurs sont très simples. Le client va gérer les interactions avec les ressources sur un serveur. Il peut, par exemple, récupérer une ressource grâce à une méthode **GET**. Il peut également écrire les données dans une ressource existante grâce à une méthode **PUT**.

Le nombre d'interactions est très limité. HTTP ou HyperText Transport Protocol Secure (HTTPS) est un moyen de mettre en oeuvre ces méthodes.

HTTP est un protocole qui peut être utilisé pour mettre en œuvre un serveur Web les principes de REST (qualifié en anglais de **RESTfull**). HTTP définit différentes méthodes permettant au client d'interagir avec les ressources sur le serveur :

- **GET** est utilisée pour récupérer la représentation d'une ressource (par exemple page Web, valeur de température d'un capteur, etc.). Par exemple, la figure 2.5 donne le format d'en-tête HTTP GET pour récupérer une page Web ;
- **HEAD** est utilisée pour récupérer uniquement les métadonnées présentes dans les en-têtes de réponse sans le corps de réponse ;
- **POST** est utilisée pour indiquer au serveur une nouvelle ressource ;
- **PUT** est utilisée pour stocker une ressource à l'endroit identifié par l'URI dans la requête. Si la ressource existe déjà, elle sera modifiée ;

- **PATCH** permet au client de ne modifier qu'une partie de la ressource ;
- **DELETE** est utilisée pour supprimer la ressource définie.

Question 2.2.2: Etat

Le serveur garde un état des précédentes requêtes ?

- Vrai
- Faux

Question 2.2.3: Wold Wide Web

Le World Wide Web est basé sur ce principe des états pour :

- fonctionner à la fois sur des ordinateurs et des téléphones portables,
- pouvoir servir un grand nombre de requêtes,
- chiffrer les communications.

Question 2.2.4: Repésentation de l'Information

Quels sont les formats qui permettent de représenter des informations structurées (2 réponses) :

- | | | | |
|-----------------------------------|-------------------------------|-------------------------------|-------------------------------|
| <input type="checkbox"/> Ethernet | <input type="checkbox"/> IPv4 | <input type="checkbox"/> MQTT | <input type="checkbox"/> JSON |
| <input type="checkbox"/> IEEE | <input type="checkbox"/> IPv6 | <input type="checkbox"/> HTTP | |
| <input type="checkbox"/> 802.15.5 | <input type="checkbox"/> UDP | <input type="checkbox"/> CoAP | |
| <input type="checkbox"/> Wi-Fi | <input type="checkbox"/> TCP | <input type="checkbox"/> XML | |

Question 2.2.5: Schéma

Dans l'URI <https://plido.net/unit/definition.html>, quel est le schéma ?

Question 2.2.6: Autorité

Dans l'URI <https://plido.net:8080/unit/definition.html>, quelle est l'autorité ?

2.3 Modèle Publish/Subscribe

Il existe d'autres formalismes que REST. Un autre formalisme, très populaire, est orienté "diffusion" en utilisant le principe "publication/abonnement" ou "**publish/subscribe**". Comme nous allons le montrer dans la suite, même si les fonctionnalités entre ces deux modes peuvent sembler similaires à HTTP, la philosophie de conception est très différente : publish/subscribe vise des applications intégrées tandis que REST vise l'interopérabilité globale.

Le modèle publish/subscribe fait le découplage entre l'expéditeur d'un message et son destinataire. Dans ce paradigme (cf. figure 2.6 page suivante), il existe des "Publishers" qui produisent des données ou des messages et envoient le message à une entité généralement appelée "Broker". En

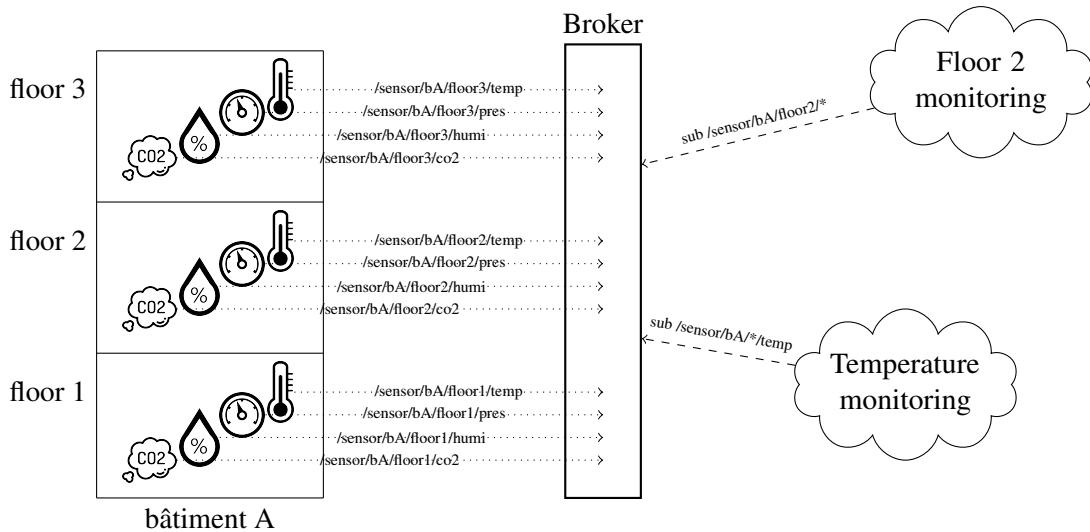


FIGURE 2.6 – Exemple de topics MQTT.

outre, les messages peuvent être classés en "Topics", contenus ou types, etc. Ensuite, il existe des abonnés qui souscrivent au broker, par exemple à un topic donné, afin de recevoir les messages qui les intéressent, comme montré dans le schéma.

Le broker peut alors utiliser des filtres pour envoyer uniquement ces messages aux abonnés du topic concerné. Il existe plusieurs protocoles Publish-Subscribe tels que Message Queuing Telemetry Transport (MQTT), Advanced Message Queuing Protocol (AMQP), Java Messaging Service (JMS) ou Extensible Messaging Protocol et Presence (XMPP).

2.3.1 MQTT

MQTT est détaillé dans la suite du cours car il est très populaire pour la communication entre processus, mais également dans l'internet des objets.

Développé initialement par IBM en 1999, pour le middleware MQSeries, il est devenu une norme OASIS en 2013, puis en 2016 une norme ISO¹.

Imaginons par exemple que plusieurs capteurs soient installés, sur plusieurs étages, dans deux bâtiments A et B. Certains capteurs collectent des informations sur la température et d'autres collectent des informations sur l'humidité. Ces capteurs peuvent envoyer les données régulièrement à un broker central.

Les données peuvent être classées en différentes rubriques qui peuvent également être organisées de manière hiérarchique. Par exemple, le topic `/sensor` signifie "toutes les données de capteurs", `/sensor/buildingA/` signifie "des données de capteurs uniquement installées dans le bâtiment A". En plus, `/sensor/buildingA/floor3/temperature` pourrait signifier "des données de capteurs de température installés uniquement au troisième étage dans le bâtiment A".

Certains abonnés peuvent s'abonner aux messages en fonction de leur intérêt. Ainsi, un abonné intéressé uniquement par les données d'humidité de l'ensemble du bâtiment B peut s'abonner au sujet `/sensor/buildingB /*/humidity` et le broker n'enverra que ces données à cet abonné.

1. <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/errata01/os/mqtt-v3.1.1-errata01-os-complete.pdf>

2.3.2 différence avec REST

Les principaux avantages du paradigme publish-subscribe par rapport au paradigme client-serveur, tels qu'inclus en REST, sont les suivants :

- faible couplage entre émetteur et récepteur, le broker sert d'intermédiaire et stocke les informations ;
- passage à l'échelle. Les données provenant d'une source ne sont émises qu'une fois par la source. Le broker les recopie vers tous les abonnés. Dans un mode client/serveur, les données doivent être émises par le serveur autant que fois que les clients le demandent.

L'absence de couplage entre l'expéditeur et le destinataire se fait en termes d'espace, de temps et de synchronisation. Celui qui publie les données a une tâche simplifiée. Il n'a pas à gérer ou connaître ceux qui les consomment, il n'a qu'à les envoyer au broker.

MQTT est très léger et conçu pour les périphériques de faibles puissances. Il a une très petite empreinte logicielle et est optimisé pour fonctionner dans les environnements à faible bande passante. Cela rend MQTT idéal pour les applications IoT. Malgré tout, l'usage de TCP et des très nombreux acquittements peut s'avérer lourds pour les équipements ou les réseaux très contraints. Une version plus légère basée sur UDP existe pour ces cas d'usage, mais elle est peu utilisée.

S'ils se ressemblent, les principes de nommage des topics MQTT et des URI REST sont complètement différents. Par rapport à MQTT, le chemin dans l'URI n'a pas de sémantique. Il a juste vocation à être unique. Il ne peut pas être utilisé pour agréger plusieurs sources d'information. Si deux capteurs publient respectivement sur les topics `/sensor/buildingA/temperature` et `/sensor/buildingB/temperature`, un subscriber peut s'abonner au topic `/sensor/*/temperature` pour recevoir toutes les mesures ; ce qui est impossible avec REST : il faudra autant de requêtes que de capteurs pour récupérer l'ensemble des mesures.

Les URI sont simplement uniques au monde par leur construction alors que les topics du MQTT sont spécifiques à une application. Un topic MQTT peut être interprété différemment par deux applications différentes. Cela ne permet pas une interopérabilité sémantique. Les abonnés doivent être construits avec une connaissance des topics utilisés par les publieurs.



3. Wireshark

Wireshark va être notre ami dans la suite de cet ouvrage pour comprendre le fonctionnement des protocoles et analyser les données qui vont circuler. Malheureusement, dans certains cas, nous devons avoir recours à des outils plus rustiques comme des traces en **hexadécimal**¹ (base 16). Il faut donc se familiariser avec ces outils, ce que nous allons faire dare-dare en analysant des requêtes HTTP simples. Si vous avez accès à un ordinateur pouvant faire tourner Wireshark, nous vous recommandons d'essayer de faire les manipulations indiquées et de répondre aux questions.

3.1 Installation

L'installation de Wireshark se fait en allant sur le site éponyme <https://www.wireshark.org/>, soit sous Linux en installant le paquetage `wireshark`. Ce programme nécessite des droits particuliers pour accéder aux messages venant du réseau, il faut les accorder au moment de l'installation.

3.2 Démarrage

Si vous lancez Wireshark avec les bons priviléges, la fenêtre d'accueil va afficher les interfaces disponibles, comme le montre la figure 3.1 page suivante sur Windows. En regard avec le modèle de référence de l'ISO, il s'agit des protocoles de niveau 2 présent sur l'ordinateur. Il peut s'agir d'une carte physique comme Ethernet ou Wi-Fi ou d'interfaces virtuelles utilisées pour communiquer en interne sur l'ordinateur.

Il s'agit de déterminer quelle interface choisir. Ce n'est pas toujours facile car leurs noms ne sont pas toujours très explicites. Les petites courbes à gauche du nom indiquent le trafic instantané que Wireshark mesure. Sur le schéma, 3 interfaces sont actives : Ethernet, la communication avec une machine virtuelle et une interface appelée **loopback**. La première permet d'avoir les communications avec l'extérieur et la dernière sera très utile lors des échanges entre deux processus dans cette machine.

1. https://fr.wikipedia.org/wiki/Système_hexadécimal

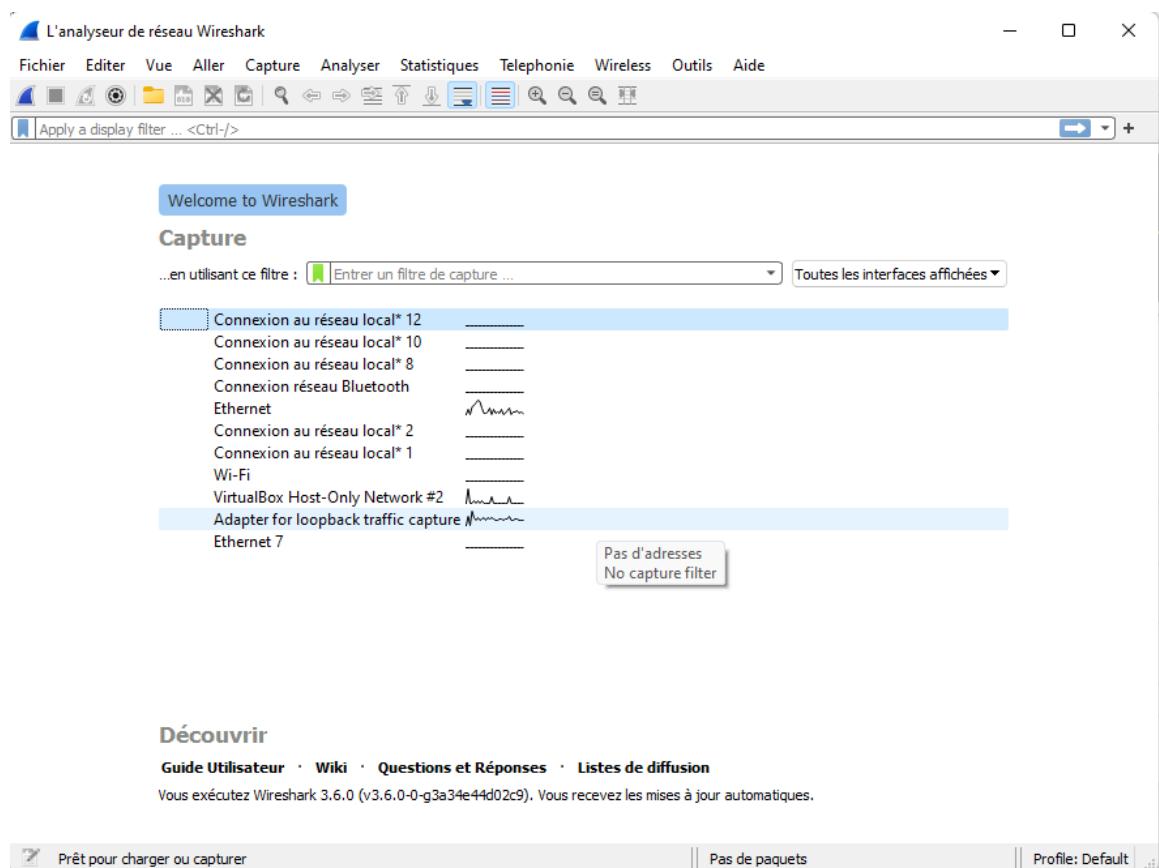


FIGURE 3.1 – Ouverture de Wireshark

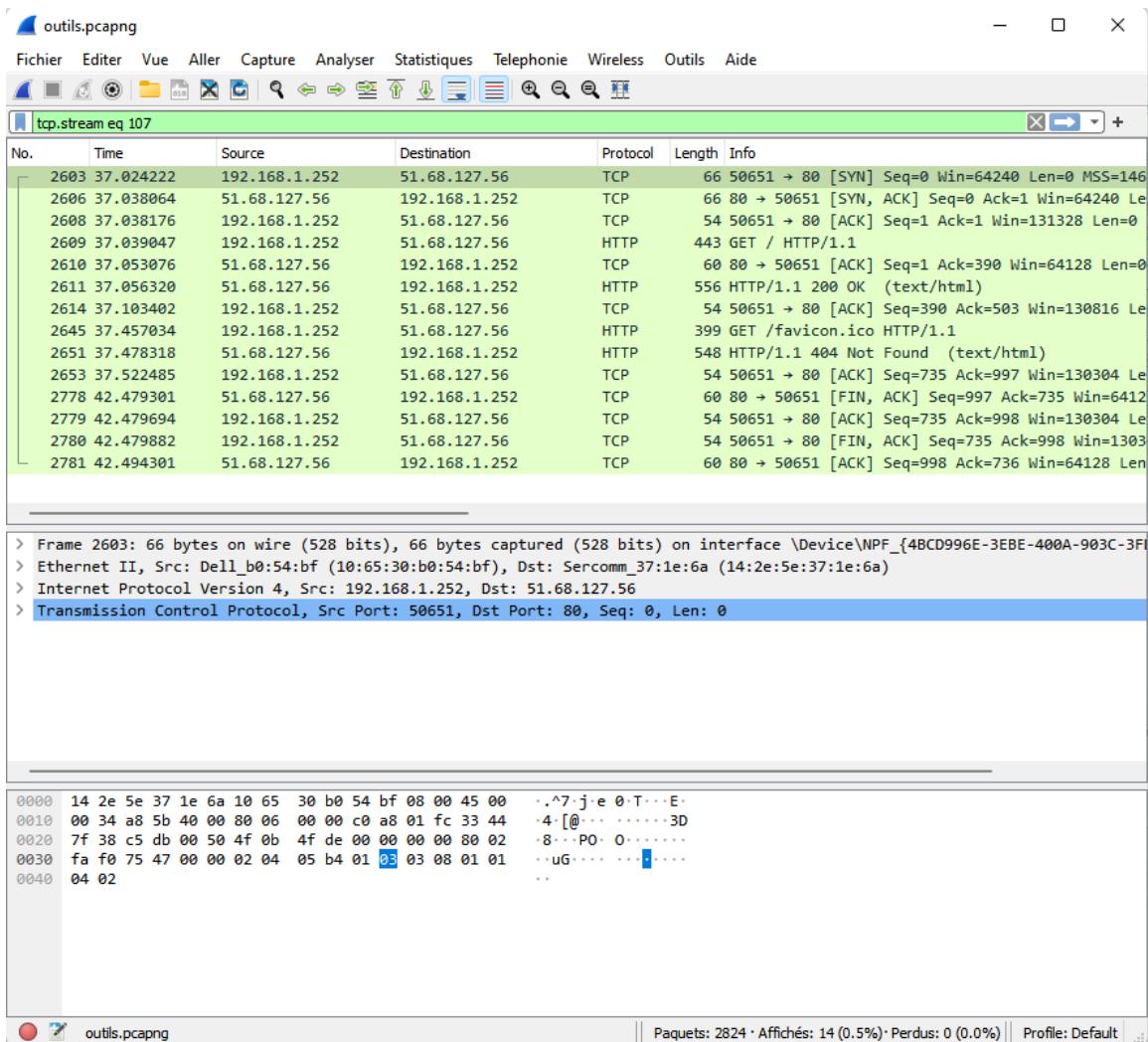


FIGURE 3.2 – Capture du trafic

3.3 Capture

En cliquant sur le nom de l'interface donnant accès au réseau extérieur (**Ethernet** dans notre cas), la fenêtre se découpe en 3 parties, comme le montre la figure 3.2.

L'écran de Wireshark se divise en 3 parties :

- en haut, défile les trames qui sont capturées sur le réseau, chaque protocole à une couleur dédiée pour faciliter le repérage :
 - le numéro de trame capturée, il s'agit d'une information ajoutée par Wireshark,
 - l'heure de capture de la trame. Cette information est aussi ajoutée par Wireshark,
 - l'adresse IP (IPv4 ou IPv6) de la machine à l'origine du paquet,
 - l'adresse IP (IPv4 ou IPv6) de la machine destinataire du paquet,
 - le protocole de plus haut niveau contenu dans la trame. Dans notre cas, cela peut être TCP si le message TCP ne contient pas de données, comme lors de l'ouverture de connexion, ou de certains acquittements. On voit également les messages HTTP qui sont

- bien entendu encapsulés dans TCP,
- la taille en octets de la trame capturée par Wireshark,
 - finalement Wireshark fournit un résumé du contenu de la trame, pour comprendre ce qui se passe sur le réseau. Dans la capture, on retrouve pour les messages HTTP, les requêtes GET ou les notifications ;
 - si une trame est sélectionnée dans la liste, elle apparaît dans la zone du milieu avec l'empilement protocolaire. Le contenu de chacun de ces protocoles peut être détaillé en cliquant sur le petit triangle à gauche ;
 - la fenêtre du bas donne l'équivalent en hexadécimal. Les parties surlignées correspondent aux champs sélectionnés dans la fenêtre du milieu. À noter que l'on retrouve l'information à la fois en hexadécimal et en caractère American Standard Code for Information Interchange (ASCII), ce qui aide à la lecture quand on cherche une valeur spécifique.

Question 3.3.1: Première colonne

Dans la première colonne :

- Le numéro de la trame attribué par Wireshark à sa réception
- Le numéro de la trame relevé directement dans la trame Ethernet

Question 3.3.2: Deuxième colonne

Dans la deuxième colonne :

- L'heure de réception par Wireshark
- L'instant d'émission de la trame

Question 3.3.3: Les troisième et quatrième colonnes

Dans les troisième et quatrième colonnes :

- Les adresses Ethernet des machines.
- Uniquement les adresses IPv4 des machines.
- Les adresses IPv4 ou IPv6 des machines.

Question 3.3.4: La cinquième colonne

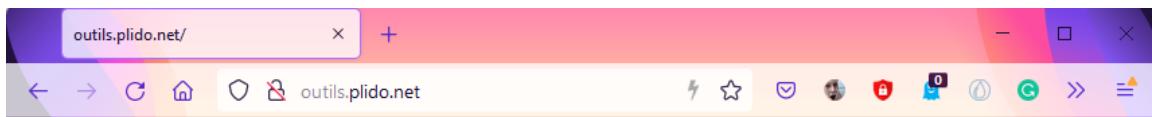
Dans la cinquième colonne :

- Le protocole applicatif (niveau 7).
- Le dernier (de plus haut niveau) protocole reconnu.
- Le protocole de niveau 4 (ici TCP ou UDP).

Question 3.3.5: La sixième colonne

Dans la sixième colonne :

- La taille en bits de la trame.
- La taille en octets de la trame.



Hello!

This is the default web page for this server.

It just display a simple page. You will find a more sophisticated page [here](#)

FIGURE 3.3 – Affichage de la page par Firefox

FIGURE 3.4 – Display of the page by Firefox

Question 3.3.6: La septième colonne

Dans la septième colonne :

- Un résumé des informations transportées par le protocole de plus haut niveau.
- Les options d'IPv4.
- Le contenu en ASCII du message de plus haut niveau.

Cela fait beaucoup de trafic, nous allons limiter ce qui est affiché en ajoutant un filtre à un destinataire particulier. Le site `outils.plido.net` à l'adresse IPv4 `51.68.127.56`. Dans la fenêtre où il est indiqué *Apply a display filter*: taper les instructions suivante :

```
ip.addr==51.68.127.56
```

n'oubliez pas le double == et la fenêtre doit devenir verte quand tout sera tapé indiquant que la syntaxe du filtre est correcte. En appuyant sur entrée, la fenêtre doit se vider.

3.3.1 Analyse du trafic web

Dans la barre d'adresse de votre navigateur préféré, taper l'URL suivante :

```
http://outils.plido.net
```

et la page Web indiqué figure 3.4 doit apparaître.

Wireshark a permis de visualiser le trafic échangé entre l'ordinateur et le serveur Web. Le trafic doit être similaire à celui de la figure 3.2 page 39. La figure s'obtient en sélectionnant le menu *Statistiques/Graphique de flux* et en cochant *Limiter au Filtre d'Affichage*. Elle est un peu plus lisible car elle représente les échanges sous forme de chronographes.

Trois phases peuvent être distinguées :

- L'ouverture de connexion TCP avec l'émission de trois messages TCP ;
- la phase de transfert de données :
 - le client envoie une requête HTTP GET au serveur pour demander la ressource à la racine (/),

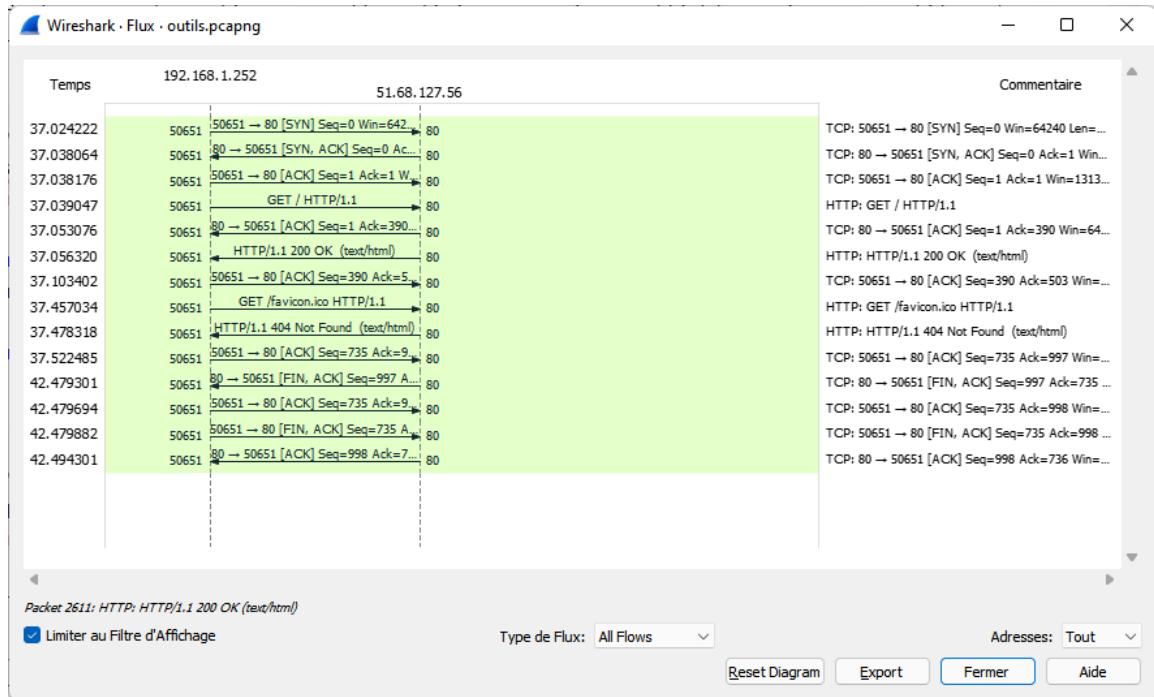


FIGURE 3.5 – Diagramme temporel des échanges.

- le serveur acquitte le message au niveau TCP pour indiquer qu'il a bien été reçu.
- le serveur envoie la réponse à la requête précédente et précisant le statut (200 : OK) et le contenu est formaté en HTML.
- le client acquitte ce message au niveau TCP,
- le client envoie une nouvelle requête HTTP GET pour obtenir la ressource /favicon.ico
- le serveur répond que la ressource n'existe pas (404 : Not Found). Cette réponse acquitte implicitement le message précédent.
- le client acquitte la réponse du serveur au niveau TCP.

- le serveur termine la connexion après 5 secondes d'inactivité. La fermeture se fait en échangeant 4 messages TCP.

Question 3.3.7: Code de notification de HTTP

Dans la trace suivante, nous avons vu que le serveur répondait aux requêtes du client par un numéro à 3 chiffres. A l'aide du [RFC 7231](#), pouvez-vous attribuer le chiffre de gauche à une catégorie de notifications :

- 0
- 1
- 2
- 3
- 4
- 5

- Redirection
- Erreur coté serveur
- Erreur coté client
- Non attribué
- Succès
- Information

3.3.2 Analyse des requêtes HTTP

La version 1.1 du protocole HTTP est spécifiée par le [RFC 7230](#). Nous allons regarder une petite description en anglais de l'architecture et des formats des messages.

2. Architecture

HTTP was created for the World Wide Web (WWW) architecture and has evolved over time to support the scalability needs of a worldwide hypertext system. Much of that architecture is reflected in the terminology and syntax productions used to define HTTP.

2.1. Client/Server Messaging

HTTP is a stateless request/response protocol that operates by exchanging messages across a reliable transport- or session-layer "connection". An HTTP "client" is a program that establishes a connection to a server for the purpose of sending one or more HTTP requests. An HTTP "server" is a program that accepts connections in order to service HTTP requests by sending HTTP responses.

Question 3.3.8: Organisme de standardisation.

Quelle organisation de standardisation a publié ce document ?

- Microsoft
- ISO
- IEEE
- IETF

Le RFC indique ensuite : The RFC then states :

The terms "client" and "server" refer only to the roles that these programs perform for a particular connection. The same program might act as a client on some connections and a server on others. [...]

Most HTTP communication consists of a retrieval request (GET) for a

representation of some resource identified by a URI. In the simplest case, this might be accomplished via a single bidirectional connection (==>) between the user agent (UA) and the origin server (O).

```
request    >
UA ===== 0
          < response
```

A client sends an HTTP request to a server in the form of a request message, beginning with a request-line that includes a method, URI, and protocol version, followed by header fields containing request modifiers, client information, and representation metadata, an empty line to indicate the end of the header section, and finally a message body containing the payload body.

A server responds to a client's request by sending one or more HTTP response messages, each beginning with a status line that includes the protocol version, a success or error code, and textual reason phrase possibly followed by header fields containing server information, resource metadata, and representation metadata, an empty line to indicate the end of the header section, and finally a message body containing the payload body.

The following example illustrates a typical message exchange for a GET request on the URI "http://www.example.com/hello.txt":

Client request:

```
GET /hello.txt HTTP/1.1
User-Agent: curl/7.16.3 libcurl/7.16.3 OpenSSL/0.9.7l zlib/1.2.3
Host: www.example.com
Accept-Language: en, mi
```

Server response:

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache
Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
ETag: "34aa387-d-1568eb00"
Accept-Ranges: bytes
Content-Length: 51
Vary: Accept-Encoding
Content-Type: text/plain
```

```
Hello World! My payload includes a trailing CRLF.
```

Question 3.3.9: Formatage des messages HTTP

Est-ce que les en-têtes HTTP ont une taille fixe (vous pouvez aller voir le [RFC 7231](#) qui donne des indications sur le protocole) ?

- l'en-tête tient en une ligne de 80 caractères.
- une ligne blanche sépare l'en-tête du contenu. L'en-tête peut contenir autant de lignes que nécessaire.

Question 3.3.10: Options des en-têtes HTTP

Comment sont construites les lignes optionnelles de l'en-tête ?

- mot-clé : valeurs
- un texte non formaté
- mot-clé : longueur des données : valeurs

3.3.3 Analyse de la pile protocolaire

La trame contenant la requête HTTP GET permet de visualiser l'encapsulation protocolaire définie par le modèle de référence de l'ISO. Dans Wireshark, en cliquant sur la trame, on peut la voir désassemblée et en hexadécimal dans les deux fenêtres comme le montre la figure 3.6 page suivante.

La deuxième fenêtre montre la pile protocolaire, inversée par rapport aux représentations classique (cf. figure 2.3 page 28), mais correspondant à l'ordre des encapsulations dans la trame. Comment Wireshark a pu arriver à un tel résultat.

Ethernet

Wireshark reçoit une trame du réseau **Ethernet** ou **Wi-Fi**². Le format d'un trame Ethernet est défini par le standard **IEEE 802.3**. L'en-tête contient trois champs :

- 6 octets pour l'adresse MAC du destinataire,
- 6 octets pour l'adresse de la source,
- 2 octets pour le protocole de niveau supérieur. Ainsi la valeur 0x0800 désigne IPv4 et 0x86dd IPv6.

Suivant le principe du modèle de référence de l'ISO, les adresses sont celles des nœuds adjacents, c'est-à-dire connecté au même réseau Ethernet ou Wi-Fi.

Dans notre cas, le protocole de niveau supérieur est donc un paquet IPv4 et Wireshark peut continuer à analyser, ce qui suit. Formellement se sont des données de la trame Ethernet, mais elles peuvent être comprise comme un paquet IPv4.

Question 3.3.11: Mon adresse

Dans l'exemple, figure 3.6 page suivante, qu'elle est l'adresse Ethernet de la machine émettrice de la trame ?

2. Pour le réseau Wi-Fi, il transforme le format en celui d'une trame Ethernet pour un affichage plus compact.

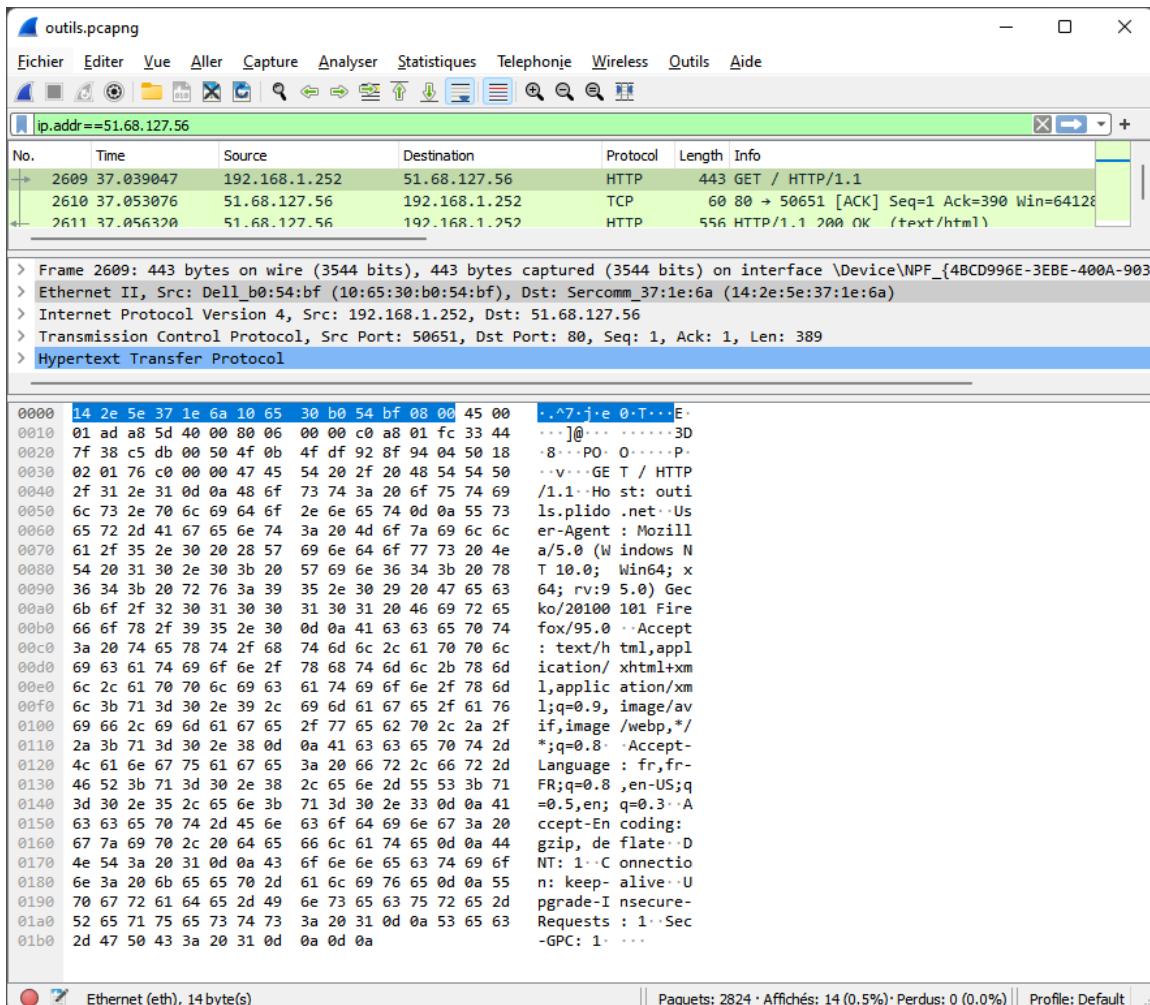


FIGURE 3.6 – Contenu de la trame transportant la requête HTTP GET

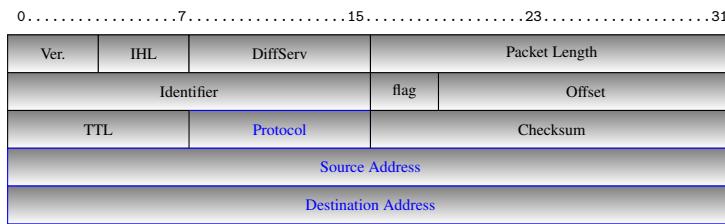


FIGURE 3.7 – Format d'un en-tête IPv4

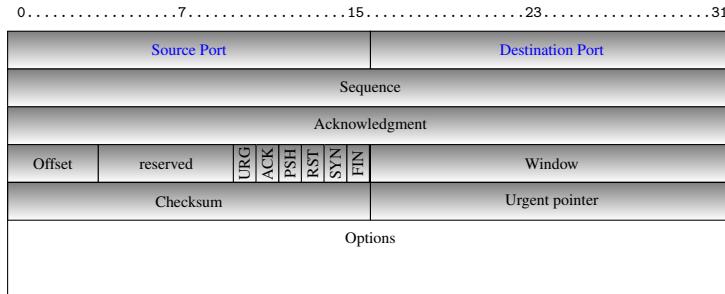


FIGURE 3.8 – Format d'un en-tête TCP

IPv4

Le format des paquets IPv4 défini dans le [RFC 791](#) a très peu évolué depuis sa publication en 1981. La figure 3.7 reprend ce format. Sans entrer dans les détails, les champs :

- adresse source et destination vont contenir les adresses IPv4 sur 32 bits des équipements d'extrémité. Des équipements intermédiaires, appelés routeur se chargent de recopier le paquet vers sa destination.
- Le champ protocole désigne la couche supérieure, la valeur 6 correspond à **TCP** et 17 à **UDP**.

Question 3.3.12: saut par saut

Est-ce que l'adresse Ethernet 14:2e:5e:37:1e:6a que l'on retrouve dans le paquet 3.6 page ci-contre correspond à l'adresse Ethernet du destinataire du paquet ? A quoi correspond elle ?

TCP

Wireshark, à partir du champ protocole valant 0x06, détermine que les données IP qui suivent l'en-tête sont un message TCP, il peut donc poursuivre le désassemblage de la trame. Le format de l'en-tête TCP est donné figure 3.8. Les numéros de port déterminent quelle application est utilisée. Si un client va prendre un numéro quelconque (50651 dans la trace figure 3.6 page ci-contre), les serveurs vont utiliser des numéros connus de tous. Ainsi, les serveur Web se vont vus attribuer la valeur 80. Ils peuvent en choisir d'autres, comme on l'a vu lors de la constructions des URL.

Wireshark connaît cette liste de numéro de port bien connu et peut continuer à analyser la trame comme étant du HTTP.

Sans entrer dans les détails, on peut aussi remarquer une série de valeurs binaires qui servent par exemple à ouvrir ou fermer une connexion TCP. Si l'on reprend la phase d'ouverture de la connexion

(cf. figure 3.5 page 42), l'ouverture de connexion se fait par :

- l'émission par le client d'un message TCP avec le bit SYN de positionné,
- l'émission par le client d'un message TCP avec le bit SYN de positionné,
- le client répond en renvoyant un message avec le bit ACK de positionné.

Ces trois messages qui ne contiennent pas de données servent à synchroniser la valeur initiale du champ **sequence** à chaque bout de la connexion.

Question 3.3.13: Fermeture de connexion

A l'aide de la figure 3.6 page 46 ou de vos captures Wireshark, quels sont les messages impliqués dans la fermeture du connexion ?

3.4 Do it yourself

Un serveur Web peut s'écrire en Python grâce au module **Flask**. Le programme `simple_server.py` permet de créer un serveur Web sur son ordinateur.

Listing 3.1 – simple_server.py

```

1 from flask import Flask
2 app = Flask("MyFirstWebServer")
3
4 @app.route('/', methods=['GET'])
5 def hello_world():
6     return "HelloWorld"
7
8 app.run(host="0.0.0.0", port=8080)

```

Ce script nécessite quelques explications :

- L'import ligne 1 inclut l'objet Flask à partir du module flask.
- A la ligne 2 une instance d'un objet Flask, c'est-à-dire un serveur web, est créée. Un nom lui est associé à des fins de débogage.
- A la ligne 4 contient la partie la plus délicate du script. @ est un décorateur qui est utilisé en python pour ajouter des propriétés à une fonction. Ici, nous associons un chemin d'URI et une méthode REST à la fonction qui est ensuite définie. De cette façon, lorsque le serveur Flash recevra une requête GET sur ce chemin d'URI, il appellera la fonction `hello_word`.
- La fonction `hello_word` retourne simplement un texte que le navigateur affichera.
- le serveur est lancé, ligne 8, en appelant la méthode `run`. Il va attendre sur toutes les interfaces (adresse joker 0.0.0.0) et sur le port 8080.

Pour lancer le serveur, vous devez d'abord installer le module Flask avec pip.

```

# pip3 install Flask
Collecting Flask
  Downloading Flask-2.0.2-py3-none-any.whl (95 kB)
    |
    | 95 kB 4.3 MB/s
Collecting Jinja2>=3.0
  Downloading Jinja2-3.0.3-py3-none-any.whl (133 kB)
...

```

Une fois le paquetage installé, il suffit d'exécuter le programme :

```
# python3.9 simple_server.py
 * Serving Flask app 'My First Web Server' (lazy loading)
 * Environment: production
WARNING: This is a development server. Do not use it in a production
deployment.
Use a production WSGI server instead.
* Debug mode: off
* Running on all addresses.
WARNING: This is a development server. Do not use it in a production
deployment.
* Running on http://192.168.1.53:8080/ (Press CTRL+C to quit)
127.0.0.1 - - [14/Dec/2021 21:06:55] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [14/Dec/2021 21:06:59] "GET /favicon.ico HTTP/1.1" 404 -
```

Question 3.4.1: loopback

Quelle URI devez vous entrer dans votre navigateur pour accéder en local à ce serveur.

Question 3.4.2: Nom du serveur

A l'aide de Wireshark, pouvez vous déterminer dans la réponse les valeurs des options HTTP Content-Type et Server.

Ne pas oublier que le trafic passe par l'interface *loopback*. Pour afficher le trafic sur un port particulier, vous pouvez utiliser le filtre `tcp.port==XXXX`. Don't forget that the traffic goes through the interface *loopback*. To display traffic on a particular port, you can use the `tcp.port==XXXX` filter.

4. Modbus

4.1 Introduction

Modbus est apparu en 1979 à une époque où l'internet n'existe pas encore ! Il est toujours très populaire dans l'industrie. A l'origine Modbus était construit sur un bus série **RS-485** qui connectait différents équipements appelé (cf. figure 4.1) :

- secondaires ou slaves et
- un primaire appelée aussi master qui gère les communications.

Chaque secondaire a un numéro unique ou adresse. Les adresses sont comprises entre 1 et 247. Le primaire n'a pas besoin d'une adresse puisque toutes les communications ont lieu avec lui.

Le primaire envoie une requête à un secondaire et le secondaire répond au primaire. Les communications directes entre deux secondaires ne sont pas possibles.

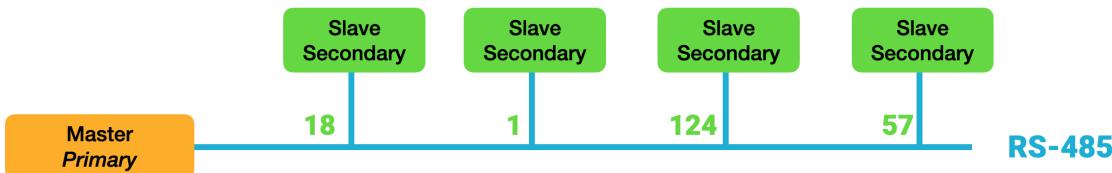


FIGURE 4.1 – Architecture filaire de Modbus



FIGURE 4.2 – Trame Modbus

4.1.1 Registres

Un équipement de Modbus peut prendre deux choses à travers des registres :

- les relais qui peuvent prendre une valeur binaire "on" ou "off". Si le primaire peut modifier l'état et, bien sûr le lire, c'est appeler un *coil*. Si la valeur binaire peut uniquement être lu c'est un *discrete input*.
- des registres sur 16 bits. Ils sont utilisés pour représenter une valeur comme un courant électrique, une température, une vitesse de rotation,... De même, si on peut uniquement lire la valeur à est appelée un *input register* sinon, si elle peut être également être modifiée par le primaire, elle est appelée un *holding register*.

Un équipement Modbus peut avoir jusqu'à 10 000 registres de ces quatre catégories.

4.1.2 Protocole

Modbus est un protocole requête/réponse. Le primaire envoie une requête à l'adresse d'un équipement pour lire ou écrire un de ces registres.

Une trame Modbus est une séquence de caractères commençant par un octet avec l'adresse du secondaire suivi d'une commande ou code de fonctions spécifique à chaque catégorie de registre :

- 1 pour lire un coil,
- 2 pour lire un discrete input,
- 3 pour lire un holding register,
- 4 pour lire un input register,
- 5 pour écrire un coil,
- 6 pour écrire un holding register.

La suite de la trame contient les données puis un Cyclic Redundancy Check (CRC) pour valider qu'il n'y a pas d'erreur de transmission dans la trame. La partie donnée peut être différente dans la requête et la réponse. Par exemple pour lire un holding register, la requête contient l'adresse du premier registre à lire et le nombre de registres à lire et la réponse contient le nombre de données transmises suivi de leurs valeurs. Pour écrire sur un registre, les données de la trame seront l'adresse du registre et les données à écrire.

4.1.3 Exemple : XY-MD02

Regardons de plus près un exemple concret. On va utiliser un capteur de température et d'humidité, le **XY-MD02** (cf. figure 4.3 page suivante) dont les spécifications sont facilement accessibles via une recherche sur Internet.

La partie verte, se compose de quatre connecteurs dont la signification est indiquée sur l'étiquette.

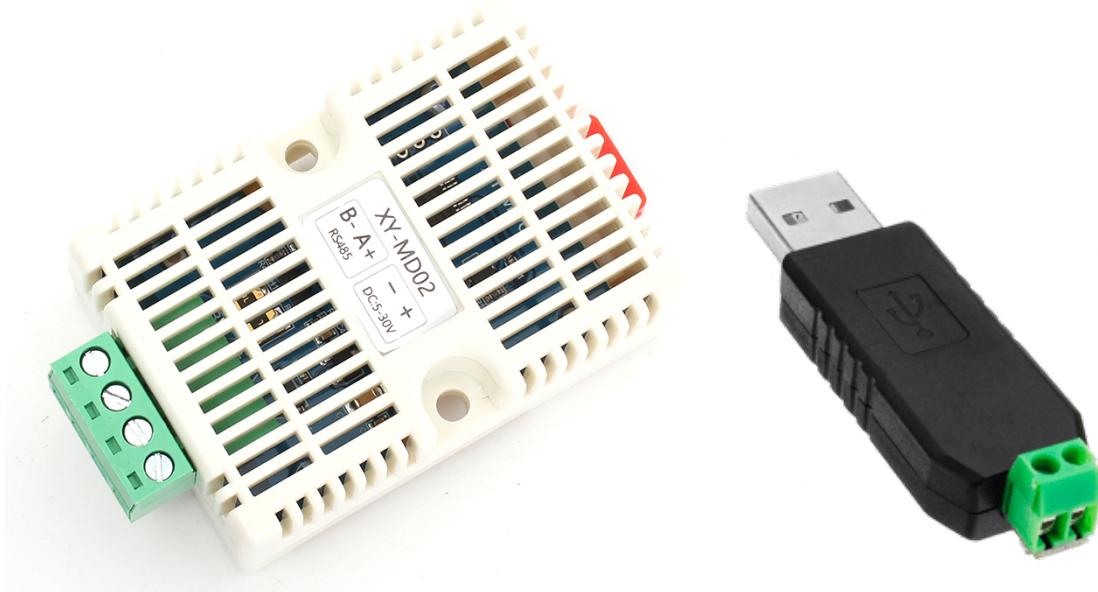


FIGURE 4.3 – XY-MD02 et Adaptateur USB/RS-485

Les deux bornes de gauche constituent le bus **RS-485** nommées A+ et B- et les deux bornes de droites permettent d’alimenter électriquement l’équipement avec une tension comprise entre 5V et 30V.

Un adaptateur **USB/RS-485** (cf. figure 4.3) est connecté à un ordinateur. On y retrouve les deux bornes A+ et B- du bus RS-485. L’ordinateur joue le rôle de primaire qui va interroger le capteur de température.

Le programme **QModMaster**¹ (cf. figure 4.4 page ci-contre) permet d’interroger ou d’écrire les registres des secondaires. Dans la fenêtre de gauche permet d’accéder aux registres des secondaires. La fenêtre de droite montre le trafic ayant circulé sur le bus RS-485.

Pour que le primaire puisse se connecter au secondaire, en plus du nom du port série (ici COM3), il faut disposer de plusieurs informations que l’on peut retrouver dans sa documentation :

- la vitesse de transmission sur le bus (ici 9 600 bit/s) et le codage des caractères transmis (ici 8 bits sans bit de parité et un bit de stop)².
- l’adresse du secondaire sur le bus.

La documentation donne également la nature des registres et leur codage. Le tableau 4.1 page suivante reprend la définition des *Input Registers*. Il s’agit de registres qui ne peuvent qu’être lus. La spécification indique que la température est stockée dans le registre 1 sur une longueur de 2 octets, soit l’intégralité de celui-ci.

La documentation indique que le secondaire a l’adresse 0x01 sur le bus RS-485. Il ne reste plus

Youtube



1. <https://sourceforge.net/projects/qmodmaster/>

2. <https://en.wikipedia.org/wiki/8-N-1>

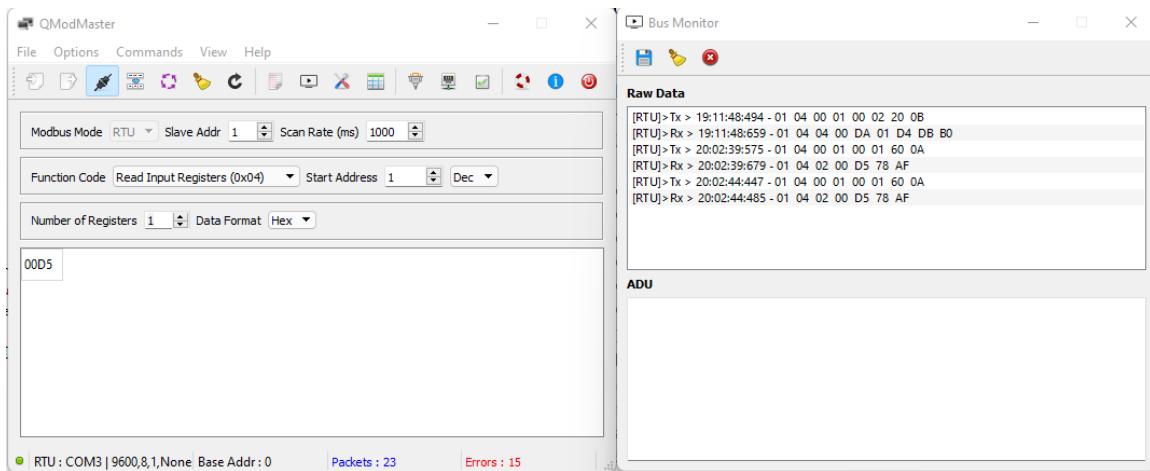


FIGURE 4.4 – QModMaster avec trace des messages

Register Type	Register Address	Register Contents	Bytes
Input register	0x0001	Temperature	2
	0x0002	Humidity	2

TABLE 4.1 – Input Register d'un XY-MD02

qu'à y envoyer une requête Modbus pour lire ce registre. La fenêtre de trace à droite sur la figure 4.4 donne les échanges. Nous allons analyser les deux dernières lignes. La première indique le contenu de la requête et la dernière la réponse du secondaire :

```
01 04 00 01 00 01 60 0A
01 04 02 00 D5 78 AF
```

La requête commence par l'adresse du secondaire (01), puis par l'action (04) pour lire un *Input Register*, suivit de l'adresse du registre (00 01) et du nombre de registres à lire (00 01). La requête se termine par la CRC validant l'intégrité de la trame (60 0A).

La réponse contient également l'adresse du secondaire (01) et l'action, suivi de la taille de la réponse en octets (02) et du résultat demandé (00 D5).

Question 4.1.1: Humidité

En regardant les échanges de la figure 4.4 quelle est la valeur mesurée pour l'humidité ?

Reste à pouvoir interpréter cette valeur. La documentation indique que la valeur est en dixièmes de degrés et que l'unité est le Celsius. En convertissant 00 D5 en décimal, on obtient 213, soit 21.3°C.

Register Type	Register Address	Register Contents	Bytes
Holding register	0x0101	Device Address	2
	0x1202	Bit rate : • 0 : 9600 • 1 : 14400 • 2 : 19200	2
	0x0103	Temperature correction -10°C - 10°C	2
	0x0104	Humidity correction -10%RH - 10%RH	2

TABLE 4.2 – Holding Register d'un XY-MD02

Question 4.1.2: Évolution des températures

En regardant les échanges de la figure 4.4 page précédente quelle est l'évolution de la température au cours du temps ?

En résumé, on voit qu'il serait très difficile de faire fonctionner l'équipement sans documentation pour connaître : la vitesse du bus RS-485, l'adresse du secondaire, les adresses des registres utilisés et leur contenu et le codage de l'information dans les registres et les unités utilisées. On a donc un degrés d'interopérabilité faible, les deux entités doivent s'accorder sur un grand nombre de paramètres.

Question 4.1.3: Taux d'HumiditéHumidity rate

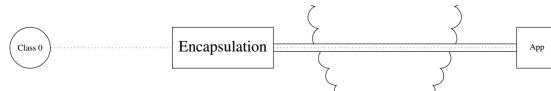
Quel est le taux d'humidité au moment de la mesure ? La documentation précise qu'il s'agit d'un pourcentage avec une précision au dizième de pourcent.

On a pu communiquer avec l'objet en utilisant les paramètres par défaut. Mais pour l'insérer dans un bus, il faut pouvoir modifier certains paramètres. La vitesse de transmission et le codage des octets doit être le même, des secondaires ne peuvent pas avoir la même addresses sur le bus.

Le XY-MD02 dispose aussi de *holding register* permettant de le paramétrage comme le montre la table 4.2

4.1.4 Passerelle IP

Il est possible d'étendre la portée d'un réseau Modbus en ajoutant une passerelle IP. Cela correspond au troisième méthode d'interconnexion de la figure 1.3 page 22. La passerelle, connectée sur le bus où restent connectés les secondaires, possède une adresse IP. Le primaire ouvre une connexion TCP avec la passerelle et y envoie ses requêtes. La passerelle recopie les données sur le bus. Inversement, les réponses des objets sont retournées à la passerelle qui les envoie au primaire en utilisation la



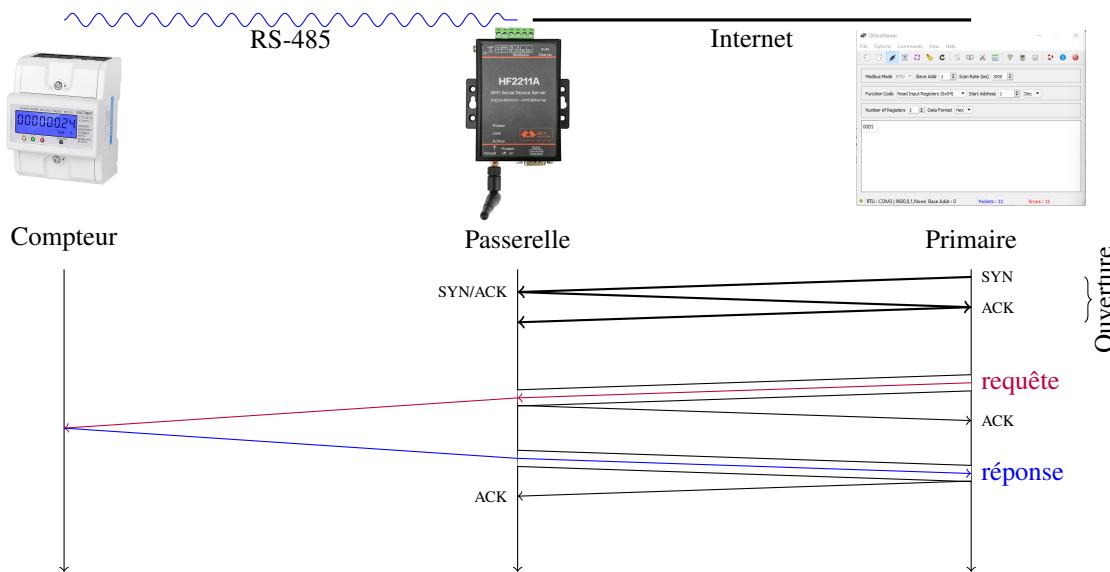


FIGURE 4.5 – Passerelle entre le réseau Internet et Modbus.

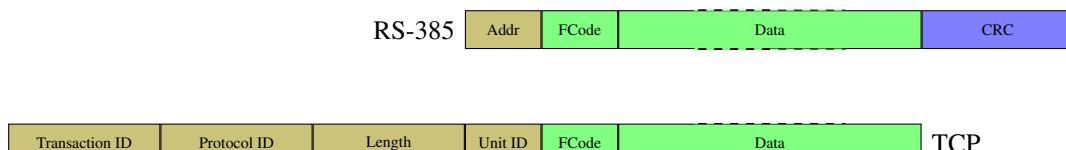


FIGURE 4.6 – Messages Modbus sur bus RS-485 et sur IP/TCP

connexion TCP. La figure 4.5 illustre les échanges. On note l’ouverture de connexion TCP qui se fait au démarrage du primaire qui reste active pour toute les échanges. On peut aussi remarquer que les messages TCP sont acquittés. La figure 4.6 montre les champs commun aux format sur le bus RS-485 et dans des paquets IP.

Comme dans l’exemple précédent du capteur de température, les spécifications du **compteur électrique** sont nécessaires pour comprendre la signification des registres utilisables. Le compteur code ses valeurs sur des nombres flottant sur 32 bits dont le codage est spécifié par la norme **IEEE 754**³. Ces valeurs sur 32 bits doivent être codées sur deux registres consécutifs d'où l'incrémentation de 2 en 2 que l'on retrouve sur la tableau 4.3 page suivante, le compteur pouvant mesurer trois phase électriques.

Wireshark peut capturer une requête ayant circulé sur le réseau Ethernet, coté primaire (cf. figure 4.7 page suivante).

0000	98 d8 63 62 29 49 10 65 30 b0 54 bf 08 00 45 00	..cb)I.e0.T...E.
0010	00 34 db ef 40 00 80 06 00 00 c0 a8 01 fc c0 a8	.4...@.....
0020	01 57 e2 c9 01 f6 16 90 37 98 5d 57 a0 fa 50 18	.W.....7.]W..P.
0030	02 01 84 ca 00 00 00 0a 00 00 00 06 1c 04 00 00
0040	00 02 ..	

3. https://en.wikipedia.org/wiki/IEEE_754

Register Type	Register Address	Register Contents	Unité	Format
Input register	0x0000	Voltage phase A	V	IEEE 754
	0x0002	Voltage phase B	V	IEEE 754
	0x0004	Voltage phase C	V	IEEE 754
	0x0008	Intensité phase A	A	IEEE 754
	0x000A	Intensité phase B	A	IEEE 754
	0x000C	Intensité phase C	A	IEEE 754
	0x0010	Puissance Totale	KWh	IEEE 754
	0x0012	Puissance phase A	KWh	IEEE 754
	0x0014	Puissance phase B	KWh	IEEE 754
	0x0016	Puissance phase C	KWh	IEEE 754
	0x0036	Fréquence	Hz	IEEE 754

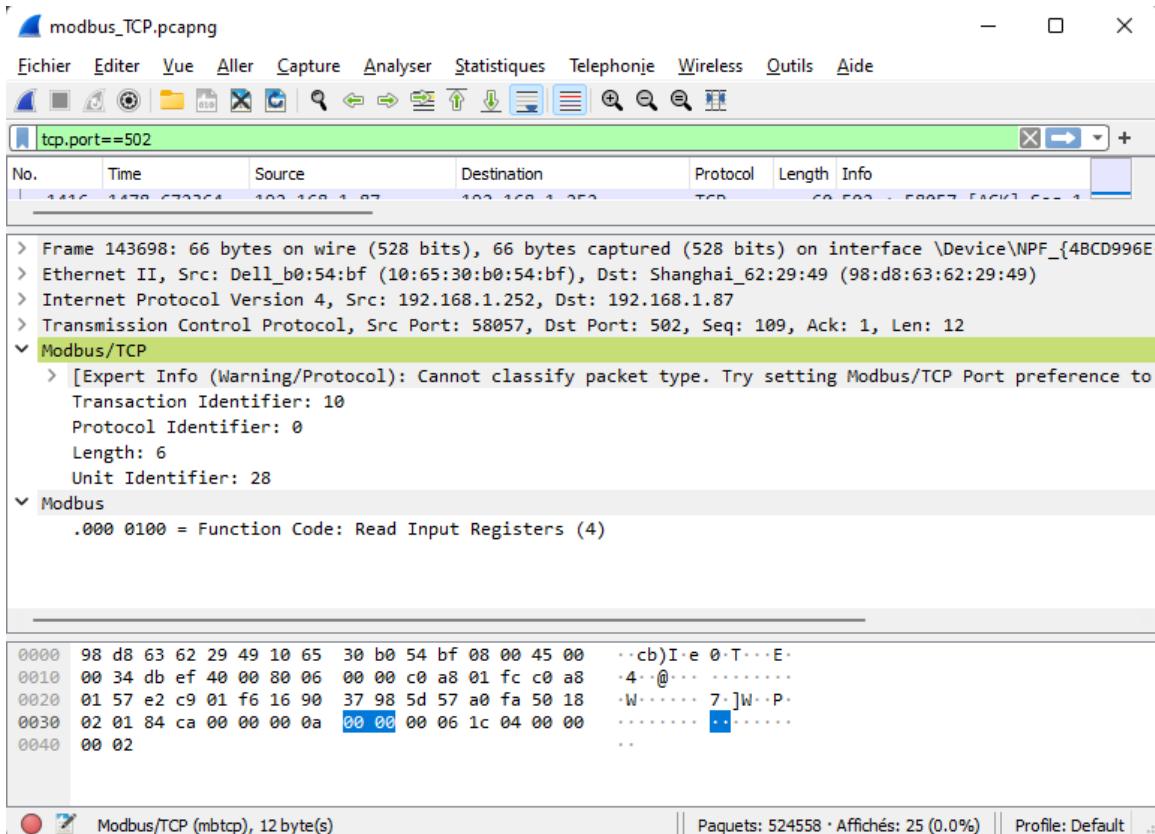
TABLE 4.3 – quelques *Input Register* du compteur électrique

FIGURE 4.7 – Capture avec Wireshark d'une trame contenant un message Modbus.

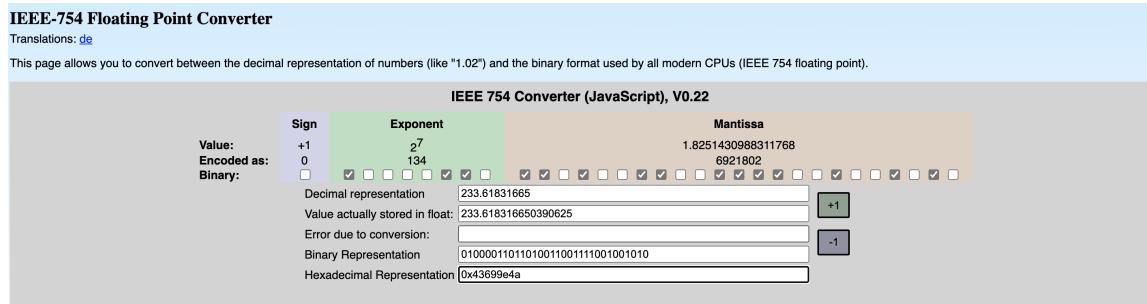


FIGURE 4.8 – Conversion d'un nombre flottant

On retrouve les encapsulations des protocoles Ethernet, IP et TCP, suivi des données TCP. Elles se composent de trois champs qui n'existent pas dans la requête circulant sur le bus RS-485 :

- le numéro de la transaction sur deux octets incrémenté à chaque requête,
- la version du protocole sur deux octets,
- la longueur en octet de la transaction.

Les champs suivants sont identiques à ceux de la trame sur le bus RS-485 :

- l'adresse du secondaire sur un octet, ici 0x1c ou 28,
- la nature de la requête : 0x04 pour lire un *input register*,
- le registre à lire sur deux octets, ici 0x0000 correspondant à la tension sur la phase A,
- le nombre de registre à lire, ici 2 pour obtenir les 32 bits de la valeur.

Le primaire reçoit la réponse suivante :

0000	10	65	30	b0	54	bf	98	d8	63	62	29	49	08	00	45	00	.e0.T...cb)I..E.
0010	00	35	c9	75	00	00	40	06	2c	aa	c0	a8	01	57	c0	a8	.5.u..@.,....W..
0020	01	fc	01	f6	e2	c9	5d	57	a0	fa	16	90	37	a4	50	18]W....7.P..
0030	44	70	e1	6e	00	00	00	0a	00	00	00	07	1c	04	04	43	Dp.n.....C
0040	69	9e	4a														i.J

On y retrouve, après les encapsulations protocolaires d'Ethernet, IP et TCP les données suivantes :

- le numéro de transaction qui correspond à celui employé dans la requête précédente. cela permet de faire le lien entre les deux messages qui aurait pu se défaire en cas de perte de paquets sur le réseau Internet,
- la version du protocole,
- la longueur de la réponse, ici 7 octets,
- l'adresse du secondaire ayant répondu, ici 28,
- la nature de la requête,
- le nombre d'octets retournés, ici 4,
- et la valeur des deux registres 0x43699e4a qui correspond à un nombre flottant tel que le représente le standard IEEE 754. Il existe de nombreux sites sur Internet qui permettent la conversion⁴. Comme le montre la figure 4.8, on obtient la valeur 233.61831665 qui correspond bien à une tension offerte par un réseau électrique.

4. <https://www.h-schmidt.net/FloatConverter/IEEE754.html>

0000	98 d8 63 62 29 49 10 65 30 b0 54 bf 08 00 45 00	..cb)I.e0.T....E.
0010	00 34 db f3 40 00 80 06 00 00 c0 a8 01 fc c0 a8	.4..@.....
0020	01 57 e2 c9 01 f6 16 90 37 b0 5d 57 a1 14 50 18	.W.....7.]W..P.
0030	02 01 84 ca 00 00 00 0c 00 00 00 06 1c 04 00 366
0040	00 02	..
0000	10 65 30 b0 54 bf 98 d8 63 62 29 49 08 00 45 00	.e0.T...cb)I..E.
0010	00 35 8b 15 00 00 40 06 6b 0a c0 a8 01 57 c0 a8	.5....@.k....W..
0020	01 fc 01 f6 e2 c9 5d 57 a1 14 16 90 37 bc 50 18]W....7.P.
0030	44 70 f1 f0 00 00 00 0c 00 00 00 07 1c 04 04 42	Dp.....B
0040	47 e9 5b	G. [

FIGURE 4.9 – Capture à étudier.

Question 4.1.4: Requête Modbus TCP

Soit l'échange donné figure 4.9 correspondant à une requête Modbus et à une réponse. Quel est le numéro de port utilisé par Modbus TCP.

Question 4.1.5: Requête Modbus TCP

En poursuivant l'analyse du trafic, quelle valeur de registre est demandée.

Question 4.1.6: Réponse Modbus TCP

En analysant le paquet suivant, comment peut-on vérifier que la réponse peut correspondre à la requête précédente.

Question 4.1.7: Réponse Modbus TCP

Quelle valeur est renvoyée. Est-ce cohérent ?

5. ARCHITECTURE POUR L'IOT

5.1 Introduction

Les objets se caractérisent par une capacité de traitement limitée et par une consommation énergétique réduite pour préserver l'autonomie imposée par une alimentation sur batterie. Or, les activités les plus consommatrices pour un équipement sont l'émission et la réception de données. Pour maximiser l'autonomie des équipements, il faut revoir l'intégralité des protocoles, mais en les calquant sur les architectures existantes pour en assurer la compatibilité.

La figure 5.1 page suivante reprend un certain nombre d'adaptation protocolaires, à différents niveau du modèle ISO, capable de s'adapter aux caractéristiques des objets contraints. Dans les chapitres suivants nous reviendrons sur ces technologies en partant de la représentation des données pour aller jusqu'aux couches basses.

5.2 Topologies

Les réseaux pour l'internet des objets peuvent être divisés en deux catégories : les topologies maillées (**Mesh** in english) et étoilées (**star**).

Réseaux Maillés

Les réseaux maillés, tels que la famille IEEE 802.15.4, sont une adaptation d'un protocole d'accès Wi-Fi pour préserver l'énergie. La portée de transmission est limitée à 50 mètres pour limiter la consommation d'énergie ; et par conséquent les messages doivent être relayés par d'autres nœuds pour atteindre leur destination.

Le débit est de quelques centaines de kilobits/s et la taille de la trame est de quelques centaines d'octets.

Ces réseaux sont performants pour transporter des données IoT, mais le protocole de routage, ainsi que le relayage des trames, consomment l'énergie des objets.

Youtube



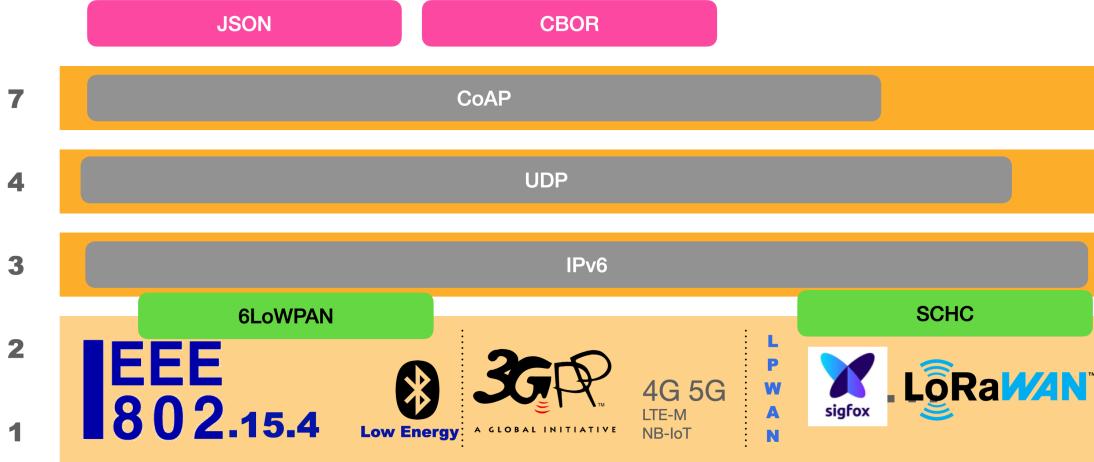


FIGURE 5.1 – Pile protocolaire de l'IoT

Réseaux en Étoile

Les topologies en **étoile** ne nécessitent pas de tels mécanismes de routage. Toutes les communications se font avec un point central qui relaie les informations vers la destination.

Les progrès réalisés dans le traitement des signaux permettent d'étendre la portée de transmission à faible puissance. Cette famille de réseaux est appelée réseaux étendus à faible puissance (Low Power Wide Area Network (LPWAN)) comme **Sigfox**, **LoRaWAN**, ou même du côté de la téléphonie cellulaire avec des évolutions de la norme **4G** et une intégration plus complète dans la **5G**. Le [RFC 8376](#) donne, en anglais, un aperçu de ces techniques.

Avec une puissance de transmission de 25 mW, il est possible de communiquer sur une distance de 3 km en milieu urbain et de 20 km dans un environnement dégagé. Les LPWAN sont compatibles avec les appareils de classe 0 car ils ne nécessitent pas la mise en place d'une pile IP. La figure ci-dessous décrit une architecture typique pour les LPWAN.

L'appareil envoie des données brutes sur le réseau radio. Le signal radio est capté par une ou plusieurs passerelles radio, et la trame est envoyée à une passerelle réseau (LoRaWAN Network Server (LNS) pour les réseaux **LoRaWAN**, et Service Capability Exposure Function (SCEF) pour les réseaux 3GPP).

Le propriétaire de l'appareil a associé l'appareil à un connecteur dans le LPWAN Network GateWay (NGW) qui peut être un URI, une adresse de broker Message Queuing Telemetry Transport (MQTT) ou une Web socket. Lorsque l'appareil envoie des données, il est relié à l'application par ce tunnel.

Certaines technologies telles que LoRaWAN ou Sigfox utilisent des bandes sans licence, imposant un cycle d'utilisation (**Duty Cycle**) de 0,1 à 10 % selon les canaux pour assurer l'équité entre les nœuds, empêchant ainsi qu'un équipement ne monopolise le canal de transmission. Comme cette restriction s'applique également à l'antenne du fournisseur, la communication entre le réseau et

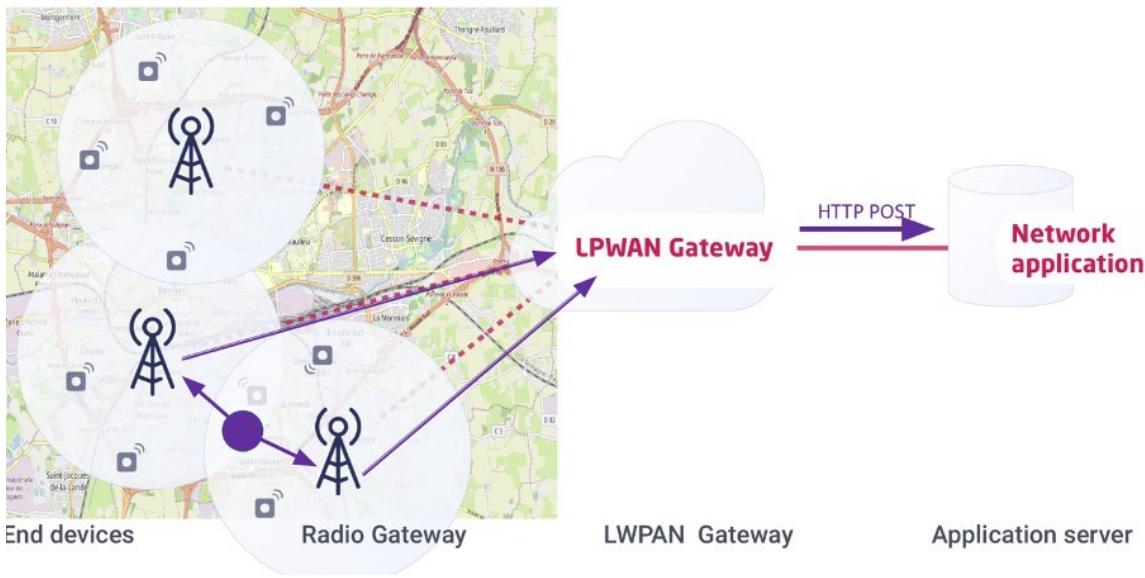


FIGURE 5.2 – Architecture simplifiée des réseaux LPWAN

l'appareil est considérablement limitée.

L'utilisation principale de ces réseaux LPWAN est la télémétrie où un appareil envoie régulièrement des informations ou une alarme de temps en temps (par exemple des capteurs de température). Le débit et la taille des messages est beaucoup plus réduit que dans le cas de réseaux maillés. The main use of these LPWAN networks is telemetry where a device regularly sends information or an alarm from time to time (for example temperature sensors). The throughput and the size of the messages is much smaller than in the case of mesh networks.

5.3 Niveaux 1 et 2

Concernant le niveau 2, le but est de gagner en énergie lors des transmissions. Déjà on peut dire adieu à Ethernet car cela imposerait d'utiliser l'infrastructure filaire et donc on ne pourrait pas placer les objets où on veut, surtout s'ils se déplacent. Les communications par ondes radio sont privilégiées.

Pour l'Internet des objets, le **Wi-Fi** est également trop gourmand en énergie. On lui préfère donc une évolution appelée **IEEE 802.15.4** qui reprend son principe de fonctionnement mais l'adapte à un faible débit et à des trames de petite taille. En particulier pour économiser l'énergie, la portée est réduite à une dizaine de mètres et il faut généralement utiliser des relais pour atteindre une destination.

Bluetooth a été adapté pour des objets avec une basse consommation Bluetooth Low Energy (BLE).

Côté téléphonie cellulaire, les protocoles évoluent pour prendre en compte les objets. La norme 4G a intégrée les communications à plus bas débit. La 5G inclura une classe permettant des communications avec les objets économies en énergie et réduisant les temps de latence.

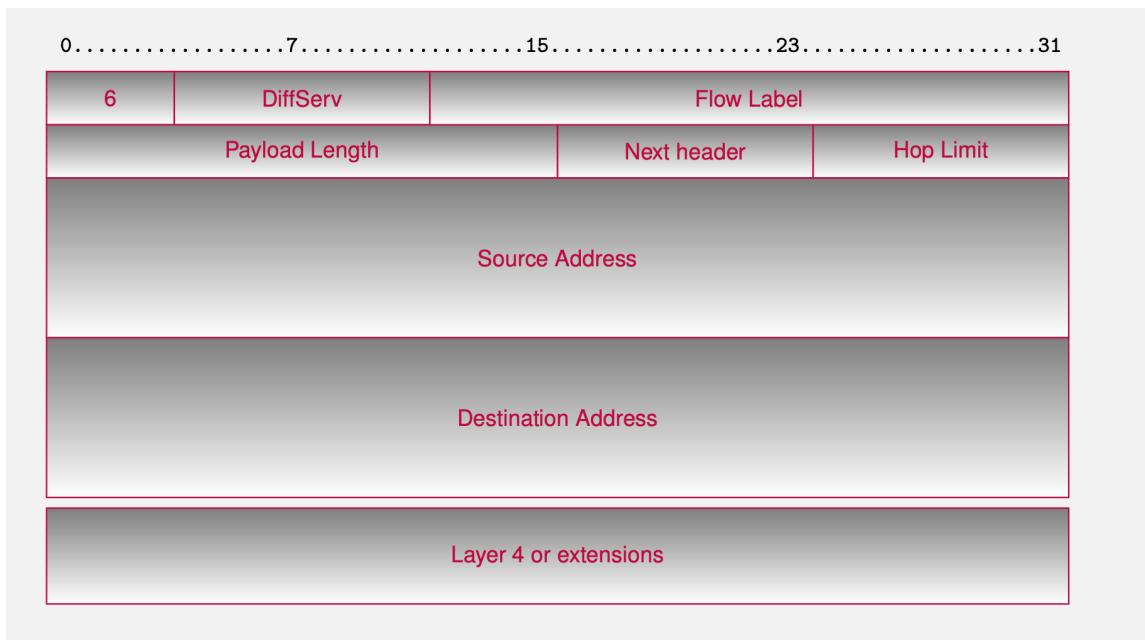


FIGURE 5.3 – Format d'un en-tête IPv6

5.4 IP et couches d'adaptation

At layer 3, we go towards the more massive use of IPv6 since version 4 has its address space saturated. The size of the address is extended on 128 bits offering 2^{96} times more addresses.

Mais, comme on le voir sur la figure 5.3 IPv6 implique des en-têtes plus grandes, ce qui est gênant car les réseaux de niveau 2 transportent de plus petites trames. Une **couche d'adaptation** entre la couche IP et le niveau 2 est nécessaire puisque les niveaux 2 conçus pour l'internet des objets ne peuvent pas transporter naturellement de grands paquets. Deux actions sont mises en œuvre : **compression** de la taille des en-têtes pour réduire leur impact, et **fragmentation** pour découper le paquet en petites trames si la première mesure ne suffit pas.

Il existe deux grandes familles de couche d'adaptation :

- **6LoWPAN** [RFC 4944](#), [RFC 6282](#), qui va intégrer un mécanisme de compression de l'en-tête IPv6 et de fragmentation pour envoyer un gros paquet divisé en petites trames. En effet, dans un réseau maillé, il n'est pas possible de se priver d'informations fournies par la couche IP car les nœuds intermédiaires en ont besoin pour acheminer le message vers le destinataire. 6LoWPAN est sans état et compresse toutes les en-têtes IPv6 sans configuration.
- Static Context Header Compression (SCHC) (prononcer chic) [RFC 8724](#) va imposer des règles décrivant l'en-tête du message et va envoyer le numéro de la règle en remplacement de l'en-tête. La compression est beaucoup plus importante et peut porter sur plusieurs couches protocolaires. Cependant, pour la mettre en œuvre, il faut avoir une idée des flux qui vont circuler sur le réseau. SCHC est spécifié pour les réseaux en **étoile** et plus particulièrement les LPWAN.

5.5 Mise en œuvre de REST

Au-dessus on avait vu que comme HTTP était le protocole dominant, TCP l'était aussi. Mais pour l'IoT ce n'est pas optimal. En effet TCP/HTTP sont des protocoles complexes qui demandent beaucoup de mémoire. Pour réduire l'impact de la pile protocolaire, l'Internet Engineering Task Force (IETF) a défini un nouveau protocole appelé Constrained Application Protocol (CoAP) qui demande que quelques Kilo Octets pour fonctionner. CoAP repose sur UDP ce qui simplifie encore la mise en œuvre.

Pour poursuivre dans l'intégration des objets dans l'internet, le protocole CoAP [RFC 7252](#) se substitue à HTTP. Il en reprend le mécanisme de nommage, d'utilisation des ressources, et les primitives de manipulation entre un client et un serveur.

La capacité de traitement du capteur et son alimentation en énergie sont souvent très limitées. La grande force de CoAP est d'être :

- facile à mettre en œuvre. Les mises en œuvre de CoAP nécessitent peu de mémoire ;
- fully compatible with HTTP and it is possible to go from one protocol to the other through generic gateways, i.e. not linked to a particular use (as shown in figure 1.3 page 22).

De ce fait, CoAP va manipuler des ressources, identifiées par des URI. Il est donc possible d'ancrer les données fournies par les objets dans l'écosystème actuel des communications entre ordinateurs, fortement structuré autour des principes REST.

La sécurité, en particulier le chiffrement des données, suit aussi les mêmes chemins que l'internet traditionnel. Il existe un chiffrement au-dessus d'UDP qui, à l'instar de HTTPS, chiffre les échanges.

5.6 Représentation des données

Pour la structuration des données, XML n'est pas utilisée car il est trop bavard. JSON est beaucoup plus efficace pour transporter des informations structurées. Il existe un équivalent binaire que nous verrons par la suite Concise Binaire Object Representation (CBOR) qui est beaucoup plus performant, et simple à mettre en œuvre est compatible avec JSON.

5.7 Alternatives à REST

Il n'est pas obligé de tout mettre en œuvre tous les protocoles définis par l'IETF. Il est possible également d'y intégrer des protocoles spécifiés pour un métier.

Par exemple, le compteur électrique **Linky** que tous les Français connaissent en implémentent qu'une partie. Au lieu d'utiliser CoAP, les électriciens utilisent leurs propres applications suivant la norme Device Language Message Specification (DLMS)/Companion Specification for Energy Management (Cosem). Celle-ci repose sur UDP puis IPv6 et **6LoWPAN** et finalement sur une variante de **IEEE 802.15.4** adaptée pour transporter l'information sur les câbles électriques.



6. LA REPRESENTATION DE LA DONNEE

6.1 Introduction

Envoyer une donnée sur un réseau n'est pas aussi simple que l'on croit.

Il faut faire la différence entre le format utilisé pour stocker des données dans la mémoire de l'ordinateur et celui employé pour l'envoyer à une autre machine. En effet, chaque machine à sa propre représentation souvent liée aux capacités de leur processeur. Cela est surtout vrai pour les nombres. Ils peuvent être stockés sur un nombre de bits plus ou moins important ou peuvent être représentés en mémoire de manière optimisée pour accélérer leur traitement.

En revanche, la représentation des chaînes de caractères (non accentués) est relativement uniforme car elle se base sur le code ASCII qui est le même pour tous les ordinateurs. Un texte de base est facilement compréhensible par toutes les machines. Une solution serait donc de n'utiliser que des chaînes de caractères.

Par exemple, si l'on veut envoyer l'entier ayant pour valeur 123, il existe plusieurs représentations possibles :

- envoyer une chaîne de caractères "123" contenant les chiffres du nombre ;
- envoyer la valeur binaire 1111011.

On voit que juste pour transmettre une simple valeur stockée dans la mémoire d'un ordinateur, il existe plusieurs options et évidemment pour que cette valeur soit interprétée de la bonne façon, il faut que les deux extrémités se soient mises d'accord sur une représentation.

Quand on veut transmettre plusieurs valeurs, c'est-à-dire quand on a des données structurées, d'autres problèmes surviennent.

Par exemple : quelle est la taille des blocs que l'on va transmettre ? Comment indiquer la fin de la transmission ? Pour une chaîne de caractères, comment indiquer qu'elle se termine ? Autre exemple :

Youtube



si l'on veut transmettre "12" puis "3", comment faire pour que l'autre extrémité ne comprenne pas "123" ?

Pour que la transmission se fasse correctement, il faut que l'émetteur et le récepteur adoptent les mêmes conventions. Quand il s'agit d'un ensemble de données, il faut être capable de les séparer. Avec les tableurs, une première méthode est possible avec la notation Comma Separated Values (CSV) Comme son nom l'indique, les valeurs sont séparées par des virgules. Les valeurs sont représentées par des chaînes de caractères. Les textes sont différenciés des valeurs numériques, par l'utilisation de guillemets. Ainsi, 123 sera interprété comme un nombre et "123" comme un texte.

Si cette représentation est adaptée aux tableurs, elle est relativement pauvre car elle ne permet de représenter que des valeurs sur des lignes et des colonnes. Pour les usages du Web, il a fallu trouver un format plus souple permettant de représenter des structures de données complexes. Évidemment, comme rien n'est simple, il en existe plusieurs et les applications échangeant des données devront utiliser le même.

On voit que l'envoi de la chaîne de caractères ne suffit pas, il faut la formater pour que le récepteur puisse trouver le type de la donnée transmise, qu'un nombre ne soit pas interprété comme une chaîne de caractères, qu'une chaîne de caractères reste une chaîne de caractères même si elle ne contient que des chiffres.

6.2 La sérialisation

Sous ce nom barbare se cache la méthode utilisée pour transmettre des données d'un ordinateur à un autre. Une donnée peut être simple (un nombre, un texte) ou plus complexe (un tableau, une structure...). Elle est stockée dans la mémoire de l'ordinateur suivant une représentation qui lui est propre. Par exemple, la taille des entiers peut varier d'une technologie de processeur à une autre, l'ordre des octets dans un nombre peut aussi être différente (little et big endian). Pour des structures complexes comme les tableaux, les éléments peuvent être rangés à différents emplacements de la mémoire.

La sérialisation consiste à transformer une structure de données en une séquence qui pourra être transmise sur le réseau, stockée dans un fichier ou une base de données. L'opération inverse, consistant à reconstruire localement une structure de données, s'appelle désérialisation.

Il existe plusieurs formats pour sérialiser les données. Ils peuvent être binaires mais ceux généralement utilisés sont basés sur des chaînes de caractères. En effet, la représentation ASCII définissant les caractères de base et codée sur 7 bits est commune à l'ensemble des ordinateurs. L'autre avantage du code ASCII est qu'il est facilement lisible et simplifie la mise au point des programmes.

Wikipédia donne ce tableau (cf. figure 6.1 page suivante) des codes ASCII datant de 1972 (une éternité en informatique) et recolorisé par nos soins.

Les caractères en orange ne sont pas imprimables. Ils permettent de contrôler la communication des données ou de gérer l'affichage en revenant à la ligne. On les reconnaît car la séquence binaire commence par 00X XXXX. On rappelle que le code ASCII est sur 7 bits ; le bit supplémentaire (bit de parité) conduisant à 1 octet était utilisé pour détecter des erreurs de transmission. Les valeurs de 0x30 à 0x39 codent les chiffres de 0 à 9.

USASCII code chart

					0	1	2	3	4	5	6	7
					NUL	DLE	SP	0	@	P	'	p
					SOH	DC1	!	1	A	Q	a	q
					STX	DC2	"	2	B	R	b	r
					ETX	DC3	#	3	C	S	c	s
					EOT	DC4	\$	4	D	T	d	t
					ENQ	NAK	%	5	E	U	e	u
					ACK	SYN	8	6	F	V	f	v
					BEL	ETB	'	7	G	W	g	w
					BS	CAN	(8	H	X	h	x
					HT	EM)	9	I	Y	i	y
					LF	SUB	*	:	J	Z	j	z
					VT	ESC	+	;	K	[k	{
					FF	FS	.	<	L	\	l	l
					CR	GS	-	=	M]	m	}
					SO	RS	.	>	N	^	n	~
					SI	US	/	?	O	—	o	DEL
b₇	b₆	b₅	b₄	b₃	b₂	b₁	Column	Row				
b₇	b₆	b₅	b₄	b₃	b₂	b₁						
b₇	b₆	b₅	b₄	b₃	b₂	b₁						
b₇	b₆	b₅	b₄	b₃	b₂	b₁						
b₇	b₆	b₅	b₄	b₃	b₂	b₁						

FIGURE 6.1 – Codage ASCII des caractères

Hexlify

En Python, il existe le module **binascii** très pratique qui permet de convertir une séquence binaire en une chaîne de caractères ou inversement :

- `hexlify` prend un tableau d'octets et le convertit en une chaîne de caractères hexadécimaux plus lisible pour les spécialistes. Cela permet de visualiser n'importe quelle séquence de données.
- `unhexlify` fait l'inverse. Il prend une chaîne de caractères et la convertit en un tableau d'octets. Cela peut vous faciliter la programmation car, dans votre code, il est plus facile de manipuler des chaînes de caractères.

Dans la suite, nous l'utiliserons pour manipuler des identifiants. Par exemple, ce bout de code illustre l'utilisation de ces fonctions :

```
mac = lora.mac()
print ('devEUI: ', binascii.hexlify(mac))

# create an OTAA authentication parameters
app_eui = binascii.unhexlify('70 B3 D5 7E D0 03 3A E3'.replace(' ',''))
```

Comme nous le verrons par la suite, la fonction `lora.mac()` retourne un tableau d'octets. La fonction `hexlify` ligne suivante le convertit en chaîne de caractères pour un affichage plus propre.

Inversement, nous devons affecter une séquence binaire à la variable `app_eui`. Nous mettons cette séquence hexadécimale en chaîne de caractères. Les espaces offrent plus de lisibilité. Ils sont retirés par la méthode `replace` et le résultat est converti en binaire grâce à `unhexlify`

6.3 Base64

Le passage d'une séquence binaire à une chaîne de caractères ASCII en représentant les valeurs conduit à un doublement du volume. Chaque bloc de 4 bits va conduire à produire un octet correspondant au caractère d'un chiffre ou d'une lettre de A à F. Le reste des codes n'est pas utilisé.

Le codage **base64** offre un meilleur rendement en utilisant 6 bits pour coder les valeurs. Un dictionnaire fait la correspondance entre 64 valeurs et un caractère ASCII. Cependant, si l'on veut coder 4 octets, soit 32 bits, il faudra 5 blocs de 6 bits, et il y aura deux bits restants. Le symbole `=` indique que 2 bits sont ajoutés à la fin du codage. Donc, dans notre cas, il faudra ajouter deux symboles `=` comme le montre la figure ci-dessous :

On notera que pour les petites séquences, ce codage n'est pas meilleur que la transformation de la séquence hexadécimale en chaîne de caractères. Ici, il faut 8 caractères pour coder 4 octets.

Il existe beaucoup d'outils en ligne pour faire les conversions entre ces différentes représentations, comme le site www.asciitohex.com.

Module Python : base64

En Python3, le module `base64` permet de faire ces conversions. Ce module est un peu susceptible sur les types de données à utiliser.

```
1 import base64
2
3 val = b"\x01\x02\x03\x04"
4 ser = base64.b64encode(val)
5 print (ser)
6 print (ser.decode())
```

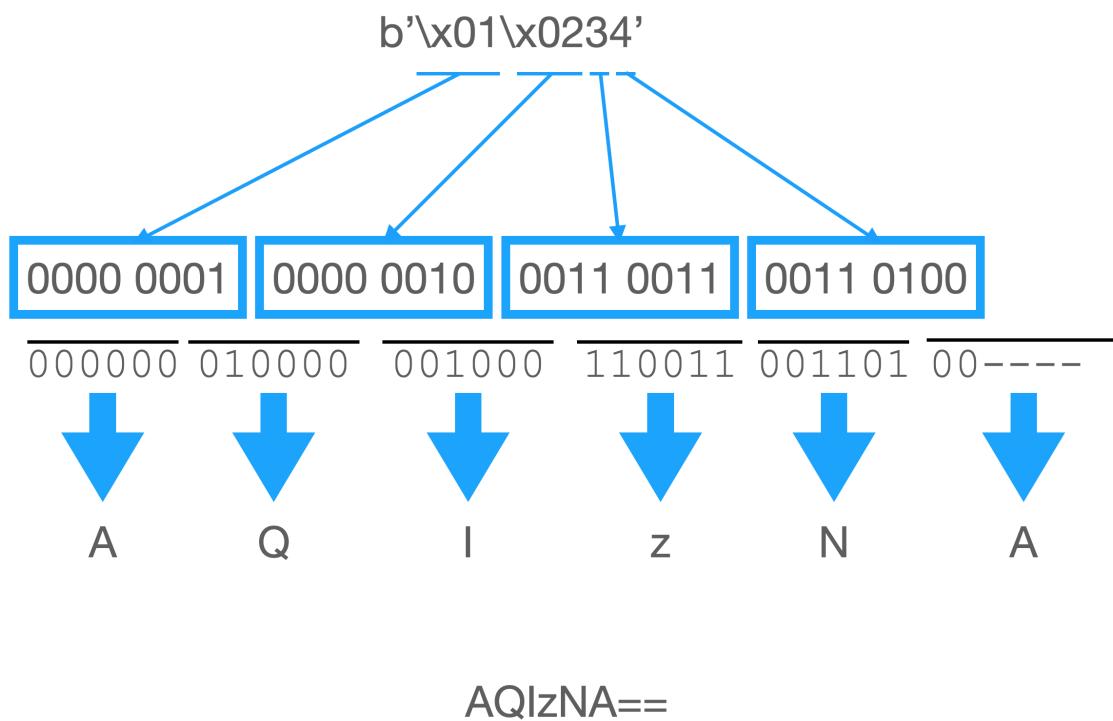


FIGURE 6.2 – Codage Base64 de données binaires

`<p>Les s´rialisations en chaînes de caractères (par exemple en Python via la commande hexlify) ou en base64 concernent surtout des donn´es binaires, mais la donn´e peut être aussi structur´e, par exemple la page d'un tableur. Il faut donc formater le document pour ´viter fusion entre les diff´rents champs. Le format CSV (Comma-Separated Values) comme son nom l'indique s´pare les donn´es par des virgules (comma en anglais). Mais si ce format s'applique bien aux donn´es d'un tableau, c'est à dire un tableau de lignes et de colonnes, il est très limit´ pour repr´enter une information telle que la mise en forme d'une page web.</p>`

FIGURE 6.3 – Codage HTML d'une page Web

```
7 ori = base64.b64decode(ser)
8 print (ori)
```

qui donne à l'exécution :

```
b'AQIzNA=='
AQIzNA==
b'\x01\x0234'
```

À noter que l'utilisation du `ser.decode()`, ligne 6, pour transformer une chaîne d'octets en chaîne de caractères, c'est-à-dire supprimer le `b` du début, peut être utilisé dans certains cas.

6.4 HTML

La sérialisation en chaînes de caractères (par exemple en Python via la commande `hexlify`) ou en Base64 concerne surtout des données binaires. Mais la donnée peut être aussi structurée, par exemple la page d'un tableur. Il faut donc formater le document pour éviter une fusion des différents champs.

HTML, sans entrer dans les détails, définit un format où les champs sont repérés par un balisage. Une balise de début est un mot clé entre `<>` et, pour une **balise** de fin, le mot clé est précédé du caractère `/`. Par exemple, la figure 6.3 avec le balisage, le premier paragraphe est formaté de cette manière dans le MOOC :

Les balises peuvent aussi prendre des arguments, comme la balise **span** dans l'exemple précédent. Ainsi, si l'on regarde une page Web, comme indiqué figure 6.4 page suivante, le navigateur est capable de l'analyser pour trouver les URI qu'elle contient. La balise `img` indiquant qu'il s'agit d'une image, le client peut interroger le serveur pour l'afficher à l'écran. Ce format structuré de sérialisation nous permet de mettre en place une caractéristique de REST, c'est-à-dire les liens entre ressources.

6.5 XML

Si HTML est dédié au formatage à l'écran de données textuelles et à la navigation sur le Web. XML¹ défini par le World Wide Web Consortium (W3C), est un format d'échange entre deux applications. Par exemple, pour échanger les notes des étudiants entre la plate-forme FUN et une autorité de certification des cours, on pourrait utiliser le format suivant :

1. <https://www.w3.org/TR/xml/>

```

<img src ="/asset-v1:MinesTelecom+04038+session01+type@asset+block/Mooc-internet_objets.jpg" width="300" type="saveimage" target="[object Object]"/>
<br>
<p><strong>Enseignants </strong><br>
<img src ="/asset-v1:MinesTelecom+04038+session01+type@asset+block/Laurent-Toutain.jpg" width="100" type="saveimage" target="[object Object]"/>
<br>Laurent Toutain<br>
</p>
</p>
<img src ="/asset-v1:MinesTelecom+04038+session01+type@asset+block/Photo_Marc_Girod_Genet.jpg" width="100" type="saveimage" target="[object Object]"/>
<br>Marc Girod-Genet<br>
</p>
</p>
<img src ="/asset-v1:MinesTelecom+04038+session01+type@asset+block/Kamal_Singh.png" width="100" type="saveimage" target="[object Object]"/>
<br>Kamal Singh<br>
</p>
</p>
<img src ="/asset-v1:MinesTelecom+04038+session01+type@asset+block/BattonHubert.jpg" width="100" type="saveimage" target="[object Object]"/>
<br>Mireille Batton Hubert<br>
<br>
<p><strong>Support pédagogique</strong>
</p>
<img src ="/asset-v1:MinesTelecom+04038+session01+type@asset+block/PatrickMaille.jpg" width="100" type="saveimage" target="[object Object]"/>
<br>Patrick Maille<br>
<br>
<p><strong>Support pédagogique</strong>
</p>
<img src ="/asset-v1:MinesTelecom+04038+session01+type@asset+block/DenisMedalSmall.png" width="100" type="saveimage" target="[object Object]"/>
<br>Denis Moalic<br>

```

FIGURE 6.4 – Capture d'une page Web

```

<etudiant>
    <prenom>John </prenom>
    <nom>Deuf </nom>
    <note>18</note>
</etudiant>

```

Il est facile en lisant l'exemple de trouver le prénom, le nom et la note de l'étudiant. On peut noter qu'il n'y a pas de différence entre la note et le nom de l'élève. Il s'agit de caractères.

S'il est syntaxiquement correct, rien ne dit que le créateur fournit quelque chose de correct qui pourra être interprété par une autre instance. XML peut inclure une grammaire ou un schéma qui est utilisé pour valider que les informations représentées dans le fichier sont non seulement syntaxiquement conformes au langage XML, mais aussi conformes au schéma. Ce schéma va décrire les champs attendus et leur type (texte, nombre...). Vous pouvez accéder à ce cours si vous voulez en savoir plus sur les schémas XML.

Du point de vue de l'internet des objets, même si le XML pourrait être un bon candidat pour l'échange d'informations, il est un format trop lourd et donc énergivore. On peut noter que pour envoyer une note sur 20 qui, dans l'absolu, prendrait 6 bits, on transmet `<note>18</note>`, soit 15 caractères soit 120 bits !

6.6 JSON

JSON offre un moyen de structurer l'information de manière plus compacte que XML. JSON s'impose comme le langage commun pour échanger les informations. A l'origine, JSON était utilisé par **Javascriptt** pour échanger des informations ; par exemple, pour afficher en temps réel l'évolution des cours de la bourse ou pour afficher des graphiques dynamiques sur l'écran de l'utilisateur.

JSON [RFC 8259](#) est un format d'échange simple. Il définit 4 types de données :



- **nombre** : Les nombres sont composés de chiffres et peuvent être positifs, négatifs, entiers ou flottants.
- **texte** : Le texte est délimité par des guillemets simples ou doubles.
- **tableau** : Les tableaux sont des listes d'éléments séparés par des virgules et entourés de crochets.
- **objet** : L'objet est une liste de paires composées d'une **clé** et d'une valeur. La clé est une chaîne de caractères et la valeur peut être de n'importe quel type. La clé doit être unique à l'intérieur d'un objet, et référence entièrement la valeur qui la suit. Le couple clé - valeur est séparé par le caractère 2 points : . Les éléments de l'objet sont séparés par des virgules. L'objet est délimité par des accolades.

Par exemple, quelques structures JSON :

- [1, -2, 0.3, 4e1] est un tableau qui contient 4 nombres ;
- [1, "2", "34"] est un tableau contenant un nombre et deux chaînes de caractères ;
- [1, [2, 3, "4"]] est un tableau de deux éléments dont le second est également un tableau de 3 éléments ;
- { "couleur" : [34, 16, 3]} est un objet qui contient un élément et la valeur est un tableau ;
- { "name" : "bob", "age" : 30} est un objet qui contient deux éléments référencés par les chaînes de caractères (ou index) "name" et "age".

L'ordre dans lequel sont placés les éléments est indifférent. {"age" : 30, "name" : "bob"} est équivalent au dernier exemple.

Cela impose que l'index utilise pour accéder à une valeur doit être unique dans la structure objet {"name" : "bob", "name" : "alice"} est interdit.

Le listing suivant donne un exemple de structure JSON tirée du [RFC 8259](#). Il contient un objet JSON avec une seule clé "Image". La valeur de cette clé est une autre structure qui contient six éléments.

```
{  
  "Image": {  
    "Width": 800,  
    "Height": 600,  
    "Title": "View from 15th Floor",  
    "Thumbnail": {  
      "Url": "http://www.example.com/image/481989943",  
      "Height": 125,  
      "Width": 100  
    },  
    "Animated": false,  
    "IDs": [11, 943, 234, 38793]  
  }  
}
```

Le balisage par clé est un élément fondamental dans la structure des données. Il est primordial d'être cohérent et d'assurer une concordance entre émetteur et récepteur sur l'intitulé de la clé pour pouvoir récupérer l'information voulue. De la même façon, il faut s'accorder sur les unités de

mesure : une interprétation d'une mesure en centimètre alors qu'elle est en pixel peut être désastreux ; c'est un problème d'interopérabilité.

JSON est facilement exploitable dans d'autres langages. Par exemple en Python, le module JSON peut être utilisé pour convertir une structure JSON qui est une chaîne ASCII en une représentation interne Python. Les tableaux sont convertis en listes et les objets en dictionnaires.

Listing 6.1 – example_json.py

```

1 import json
2 import pprint

4 struct_python = {
5     "Image": {
6         "Width": 800,
7         "Height": 600,
8         "Title": "View\u2022from\u202215th\u2022Floor",
9         "Thumbnail": {
10             "Url": "http://www.example.com/image/481989943",
11             "Height": 125,
12             "Width": 100
13         },
14         "Animated": False,
15         "Copyright": None,
16         "IDs": [0x11, 0x943, 234, 38793],
17         "Title": "Empty\u2022picture"
18     }
19 }
20
21 print(struct_python)
22 pprint.pprint(struct_python)

24 struct_json = json.dumps(struct_python)

26 print(struct_json)

28 struct_python2 = json.loads(struct_json)

30 pprint.pprint(struct_python2)

```

Le programme `example_json.py` reprend la structure précédente. La variable `struct_python` est une structure Python. On peut voir que les valeurs pour "Animated" et "Copyright" sont les mots clé Python `False` (avec un F majuscule) et `None`. Le programme affiche deux fois cette valeur avec la commande standard `print` puis avec le module `pprint` pour avoir un affichage plus lisible. On peut remarquer que l'ordre d'affichage des clés est différent. Comme "Title" était défini deux fois, seul le dernier est conservé dans la structure Python.

```

{'Image': {'IDs': [17, 2371, 234, 38793], 'Height': 600, 'Animated': False, 'Title': 'Empty picture', 'Thumbnail': {'Url': 'http://www.example.com/image/481989943', 'Width': 100, 'Height': 125}, 'Width': 800, 'Copyright': None}}
{'Image': {'Animated': False,
          'Copyright': None,
          'Height': 600,
          'IDs': [17, 2371, 234, 38793],
          'Thumbnail': {'Height': 125,
                       'Url': 'http://www.example.com/image/481989943',
                       'Width': 100},
          'Title': 'Empty picture',
          'Width': 800}}

```

Grâce à la fonction `dumps` du module `json`, la variable `struct_python` est transformée en JSON. Les mots clé `False` et `None` sont remplacés par `false` et `null`. Le programme affiche une chaîne de caractères.

```
{"Image": {"IDs": [17, 2371, 234, 38793], "Height": 600, "Animated": false, "Title": "Empty picture", "Thumbnail": {"Url": "http://www.example.com/image/481989943"}, "Width": 100, "Height": 125}, "Width": 800, "Copyright": null}}
```

Pour le retransformer, de JSON en variable Python, on utilise la fonction inverse `loads` qui traduit une chaîne de caractères en variable Python.

```
{'Image': {'Animated': False,
           'Copyright': None,
           'Height': 600,
           'IDs': [17, 2371, 234, 38793],
           'Thumbnail': {'Height': 125,
                         'Url': 'http://www.example.com/image/481989943',
                         'Width': 100},
           'Title': 'Empty picture',
           'Width': 800}}
```

Les autres langues de programmation possèdent également leur propre bibliothèque pour effectuer la traduction.

Par rapport à XML, JSON est beaucoup plus permissif et manque de formalisme pour décrire la structure. JavaScript Object Notation for Linked Data (JSON-LD) défini par le W3C renforce l’interopérabilité de JSON en introduisant des clés spécifiques décrivant la structure des données, une référence aux unités, etc. Nous verrons ces concepts dans la suite du cours.

6.7 CBOR

JSON et CBOR sont tous les deux des modes de codage de la donnée.

JSON introduit une notation très flexible permettant de représenter toutes les structures de données. Le choix de l’ASCII rend ce format universel et n’importe quel ordinateur pourra le comprendre. Mais l’utilisation de l’ASCII ne permet pas de transmettre de manière optimale l’information sur un réseau. Quand les réseaux ont un débit raisonnable, cela ne pose pas de problème. Quand on en vient à l’internet des objets, il faut prendre en compte la capacité de traitement limité des équipements et la faible taille des messages échangés.

Ainsi, en ASCII, la valeur 123 est codée sur 3 octets (un octet par caractère) tandis qu’en binaire elle n’occuperait qu’un seul octet : 0111 1011.

CBOR, défini dans le [RFC 8949](#), permet de représenter les structures de JSON mais suivant une représentation binaire. Comme nous le verrons par la suite, si CBOR est complètement compatible avec JSON, il est possible de représenter d’autres types d’information très utiles dans l’Internet des Objets.

La taille de l’information est réduite et le traitement simplifié. Il faut savoir un peu jongler avec la représentation binaire mais cela reste basique.

CBOR définit 8 types majeurs qui sont représentés par les 3 premiers bits d’une structure CBOR (cf. figure 6.5 page suivante). Ces types majeurs ont donc des valeurs comprises entre 0 et 7 (000 à 111 en binaire).

Les cinq bits suivants contiennent soit une valeur soit une longueur indiquant combien d’octets sont nécessaires pour coder la valeur. CBOR offre ainsi des optimisations qui permettent de réduire

Youtube



1 Byte



- 0 : Positive Integer**
- 1 : Negative Integer**
- 2 : Byte string**
- 3 : Text string**
- 4 : Array**
- 5 : Object list**
- 6 : Optional semantic tagging**
- 7 : Simple value and float**

FIGURE 6.5 – Définition des majeurs en CBOR

la longueur totale de la structure des données comme nous le verrons par la suite en étudiant les différents types majeurs.

6.7.1 CBOR en python

Les exemples qui vont suivre peuvent être testés sur votre ordinateur avec Python3. Si une erreur se produit au moment de la définition du module `cbor2`, vous devez l'installer sur votre ordinateur en tapant la commande :

```
# pip3 install cbor2
```

6.7.2 Type Entier Positif

JSON ne fait pas de différence entre les nombres, entiers, décimaux, positifs ou négatifs. CBOR réintroduit une distinction pour optimiser la représentation.

Le premier type majeur correspond aux entiers positifs. Il est codé par 3 bits à 0; les 5 bits suivants finissent l'octet et, suivant leur valeur, vont avoir une signification différente :

- de 0 à 23, il s'agit de la valeur de l'entier à coder;
- 24 indique que l'entier est codé sur 1 octet qui sera codé dans l'octet suivant;
- 25 indique que l'entier est codé sur 2 octets qui seront codés dans les deux octets suivants;
- 26 indique que l'entier est codé sur 4 octets qui seront codés dans les quatre octets suivants;
- 27 indique que l'entier est codé sur 8 octets qui seront codés dans les huit octets suivants.

On peut noter qu'il n'y a pas de surcoût pour coder un entier de 0 à 23. Ainsi, la valeur 15 sera

codée 0x0F (000-0 1111) tandis que, pour toutes les autres valeurs supérieures, le surcoût ne sera que d'un octet. La valeur 100 sera codé 000-1 1000² suivi du codage sur 1 octet de la valeur 100 (0110 0100).

Listing 6.2 – cbor-integer-ex1.py

```

1 import cbor2 as cbor
2
3 v = 1
4
5 for i in range (0, 19):
6     c = cbor.dumps(v)
7     print ("{:0:3}{:1:30}{:2}".format(i, v, c.hex()))
8
9     v *= 10

```

Le programme `cbor-integer-ex1.py` affiche les puissances de 10 entre 10^0 et 10^{18} :

- Ligne 1, le programme importe le module `cbor2` et le renomme pour plus de simplicité `cbor`.
- Ligne 5, la boucle permet d'avoir les multiples de 10 (variable `v`).
- Ligne 6, le module `cbor` utilise comme pour JSON la méthode `dumps` pour sérialiser une structure interne de Python dans la représentation demandée. À l'inverse, la méthode `loads` sera utilisée pour importer une structure CBOR dans une représentation interne.
- Ligne 7, le `print` permet d'aligner les données pour que l'affichage soit plus clair ; entre les accolades, le premier chiffre indique la position dans les arguments de format ; le second, après le `:`, le nombre de caractères. Par exemple, `{1:30}` indique l'argument `v` de format affiché sur 30 caractères.

Le programme donne le résultat suivant :

```

# python3 cbor-integer-ex1.py
0          1 01
1          10 0a
2          100 1864
3          1000 1903e8
4          10000 192710
5          100000 1a000186a0
6          1000000 1a000f4240
7          10000000 1a00989680
8          100000000 1a05f5e100
9          1000000000 1a3b9aca00
10         10000000000 1b0000002540be400
11         100000000000 1b000000174876e800
12         1000000000000 1b000000e8d4a51000
13         10000000000000 1b000009184e72a000
14         100000000000000 1b00005af3107a4000
15         1000000000000000 1b00038d7ea4c68000
16         10000000000000000 1b002386f26fc10000
17         100000000000000000 1b016345785d8a0000
18         1000000000000000000 1b0de0b6b3a7640000

```

2. 11000 correspond à 24

On voit facilement que les valeurs 1 et 10 sont codées sur 1 octet; que 100 est codé sur 2 octets tandis que les valeurs 1 000 et 10 000 sont codées sur 3 octets. Les valeurs entre 100 000 et 1 000 000 nécessitent 5 octets et les valeurs suivantes, 9 octets.

La taille de la représentation s'adapte à la valeur. Ainsi, il n'est pas nécessaire de définir une taille fixe pour coder une donnée.

On peut aussi noter que comme le type majeur est sur 3 bits, ce type peut être reconnu dans une lecture hexadécimale du résultat car la séquence commence toujours par le symbole 0 ou 1.

6.7.3 Type Entier Négatif

Le type majeur entier négatif est à peu près similaire à l'entier positif. Le type majeur est 001 et le codage de la valeur se fait sur la valeur absolue du nombre à laquelle on retranche 1. Cela évite deux codes différents pour les valeurs 0 et -0.

Ainsi, pour coder -15, on va coder la valeur 14, ce qui donne en binaire 001-1 1110. Ainsi, -24 peut également être codé sur 1 octet tandis que +24 sera codé sur 2 octets.

Listing 6.3 – cbor-integer-ex2.py

```

1 import cbor2 as cbor
2
3 v = -1
4
5 for i in range (0, 19):
6     c = cbor.dumps(v)
7     print ("{:0:3}{:1:30}{:2}{}".format(i, v, c.hex()))
8
9     v *= 10

```

Le programme `cbor-integer-ex2.py` reprend le même code que le programme précédent, mais la variable `v` est initialisée avec la valeur -1. Ce programme va traiter les puissances de 10 négatives.

```

# python3.5 cbor-integer-ex2.py
0          -1 20
1          -10 29
2          -100 3863
3          -1000 3903e7
4          -10000 39270f
5          -100000 3a0001869f
6          -1000000 3a000f423f
7          -10000000 3a0098967f
8          -100000000 3a05f5e0ff
9          -1000000000 3a3b9ac9ff
10         -10000000000 3b0000002540be3ff
11         -100000000000 3b000000174876e7ff
12         -1000000000000 3b000000e8d4a50fff
13         -10000000000000 3b000009184e729fff
14         -100000000000000 3b00005af3107a3fff
15         -1000000000000000 3b00038d7ea4c67fff
16         -10000000000000000 3b002386f26fc0ffff
17         -100000000000000000 3b016345785d89ffff

```

18

- 10000000000000000000000000000000 3b0de0b6b3a763ffff

6.7.4 Type Séquence binaire ou Chaîne de caractères

Les séquences binaires et les chaînes de caractères ont le même comportement. Le type majeur est respectivement 010 et 011. Il est suivi par la longueur de la séquence ou de la chaîne. Le même type de codage que pour les entiers est utilisé :

- si la longueur est inférieure à 23, elle est codée dans la suite du premier octet. On trouve ensuite le nombre d'octets ou de caractères correspondant à cette longueur ;
- si la longueur peut être codée dans 1 octet (donc inférieure à 255), la suite du premier octet contient 24 puis l'octet suivant contient la longueur suivie du nombre d'octets ou de caractères correspondant.
- si la longueur peut être codée dans 2 octets (donc inférieure à 65535), la suite du premier octet contient 25 puis l'octet suivant contient la longueur suivie du nombre d'octets ou de caractères correspondant.
- si la longueur peut être codée dans 4 octets, la suite du premier octet contient 26 puis l'octet suivant contient la longueur suivie du nombre d'octets ou de caractères correspondant.
- si la longueur peut être codée dans 8 octets, la suite du premier octet contient 27 puis l'octet suivant contient la longueur suivie du nombre d'octets ou de caractères correspondant.

Ce codage est aussi assez optimal. Il est rare d'envoyer plus de 23 caractères.

Listing 6.4 – cbor-string.py

```

1 import cbor2 as cbor
2
3 for i in range (1, 10):
4     c = cbor.dumps("LoRaWAN"*i)
5
6     print ("{0:3} {1}".format(i, c.hex()))
7
8 bs = cbor.dumps(b"\x01\x02\x03")
9 print (bs.hex())

```

Le programme `cbor-string.py` montre la représentation de chaînes de caractères de longueur croissante ainsi qu'une séquence binaire :

- ligne 3, la variable `i` prend des valeurs de 1 à 9.
- ligne 6, La multiplication d'une chaîne de caractères par un entier (ligne 4) indique le nombre de répétitions de celle-ci.
- lignes 8 et 9 montrent le codage d'une chaîne d'octets. La variable `bs` contient la représentation en CBOR d'une chaîne d'octets Python (représenté par le caractère `b` avant les guillemets, les valeurs qui ne correspondent pas à des caractères ASCII sont précédées des symboles `\x`). La représentation en hexadécimal de l'objet CBOR est ensuite affichée.

Le résultat est le suivant :

```

# python3.5 cbor-string.py
1 674c6f526157414e
2 6e4c6f526157414e4c6f526157414e
3 754c6f526157414e4c6f526157414e4c6f526157414e
4 781c4c6f526157414e4c6f526157414e4c6f526157414e4c6f526157414e
5 78234c6f526157414e4c6f526157414e4c6f526157414e4c6f526157414e4c6f526157414e
6 782a4c6f526157414e4c6f526157414e4c6f526157414e4c6f526157414e4c6f526157414e4c6f526157414e
7 78314c6f526157414e4c6f526157414e4c6f526157414e4c6f526157414e4c6f526157414e4c6f526157414e

```

Jusqu'à 3 répétitions de la chaîne de caractères "LoRaWAN", le codage de la longueur est optimal (codé sur 2 octets).

6.7.5 Type tableau

Le type tableau va regrouper un ensemble d'éléments. Chacun de ces éléments étant une structure CBOR, la seule information nécessaire pour connaître le début et la fin d'un tableau est son nombre d'éléments. Le type majeur est 100. Il existe deux méthodes pour coder la longueur d'un tableau :

- si celle-ci est connue au moment du codage, il suffit de l'indiquer avec un codage identique à celui utilisé pour indiquer la longueur d'une chaîne de caractères ;
 - si celle-ci n'est pas connue au moment du codage, il existe un code spécial pour indiquer la fin du tableau. Nous en reparlerons par la suite.

Listing 6.5 – cbor-array.py

```
1 import cbor2 as cbor
2
3 c1 = cbor.dumps([1,2,3,4])
4 print(c1.hex())
5 print()
6
7 c2 = cbor.dumps([1,[2, 3], 4])
8 print(c2.hex())
9 print()
10
11 c3 = cbor.dumps([1000, +20, -10, +100, -30, -50, 12])
12 print(c3.hex())
```

Le programme `cbor-array.py` donne quelques exemples de codage de tableau :

- [1,2,3,4] défini ligne 3, devient 8401020304. On peut deviner la structure du message CBOR : 0x84 indique un tableau de 4 éléments (attention le décodage n'est pas toujours aussi simple). Les 4 éléments sont des entiers inférieurs à 23 ;
 - [1, [2, 3], 4] défini ligne 7 devient 830182020304. Il s'agit d'un tableau de 3 éléments dont le deuxième est un tableau de deux éléments ;
 - [1000, +20, -10, +100, -30, -50, 12] défini ligne 11, devient 871903e814291864 381d38310c. On peut noter que le codage des éléments est de longueur variable, mais comme chaque élément code sa longueur, il est juste nécessaire d'en connaître le nombre.

6.7.6 Type Map (Liste de paires)

6.7.7 Type Map (List of pairs)

Le type Liste de paires ou Map est indiqué par la valeur 101. Il fonctionne de la même manière que les tableaux en comptant le nombre d'éléments. Mais cette fois-ci, la valeur représente une paire, c'est-à-dire deux objets CBOR consécutifs.

Listing 6.6 – cbor-mapped.py

```
import cbor2 as cbor  
c1 = {"type" : "hamster",
```

```

4      "taille" : 300,
5      2 : "test",
6      0x0F: 0b01110001 ,
7      2 : "program"};
8
print (cbor.dumps(c1).hex())

```

Le programme `cbor-mapped.py` donne un exemple d'encodage. A noter que la structure à encoder n'est pas directement compatible avec JSON³, certaines clés ne sont pas des chaînes de caractères.

Le résultat est `a464747970656768616d73746572667461696c6c6519012c026770726f6772616d0f1871` ce qui n'est pas très facile à lire.

cbor.me

Le site web `https://cbor.me` permet de faire automatiquement le codage dans un sens ou dans l'autre. La colonne de gauche représente la donnée en JSON et celle de droite en CBOR (dite "représentation canonique" qui facilite la lecture). En ayant entré la séquence hexadécimale ci-dessus, le site la présente comme indiqué figure 6.6 page suivante. La partie CBOR est indexée et commentée pour rendre l'objet CBOR plus lisible. Il peut également être traduit dans un équivalent JSON, bien que certaines clés restent numériques.

Sur ces exemples, on peut voir que CBOR est beaucoup plus permissif et complet que JSON, le premier champ des map CBOR peut être numérique et n'a pas à être unique dans toute la structure. Néanmoins CBOR définit un mode strict dans lequel ces clés doivent être codées en ASCII et unique pour être compatibles avec JSON. Si une clé est répété plusieurs fois dans une structure CBOR, il traiter directement l'information dans la structure CBOR et ne pas chercher à la convertir ou la déserialiser car il y a un risque de perte d'information.

Youtube



6.7.8 Type étiquette

CBOR enrichit le typage des données ; ce qui permet de manipuler plus facilement des données. Par exemple, une chaîne de caractères peut représenter une date, une URI, voire une URI codée en base 64. Le type 110 peut être suivi d'une valeur ou **tag** dont une liste exhaustive est maintenue par l'IANA⁴.

Listing 6.7 – `cbor-tag.py`

```

1 import cbor2 as cbor
2 from datetime import date, timezone
3
4 print (date.today())
5 c1 = cbor.dumps(date.today(), timezone=timezone.utc, date_as_datetime=True)
6
7 print (c1.hex())
8
9 print (cbor.loads(c1))

```

3. `json.dumps` aurait converti les clés numériques en chaînes de caractères '`{"type": "hamster", "2": "program", "taille": 300, "15": 113}`'

4. <https://www.iana.org/assignments/cbor-tags/cbor-tags.xhtml>

The screenshot shows a web-based CBOR playground at <https://cbor.me>. The interface includes a toolbar with various icons, a status bar indicating the URL and a note about RFC 8949, and a main area divided into two sections.

Left Section: Contains the JSON input: `{"type": "hamster", "taille": 300, 2: "program", 15: 113}`. Below it is a green button labeled "Diagnostic" with a dropdown arrow.

Right Section: Shows the corresponding CBOR bytes in hex format, each preceded by a byte value and a comment. The bytes are:

Byte	Comment
A4	# map(4)
64	# text(4)
74797065	# "type"
67	# text(7)
68616D73746572	# "hamster"
66	# text(6)
7461696C6C65	# "taille"
19 012C	# unsigned(300)
02	# unsigned(2)
67	# text(7)
70726F6772616D	# "program"
0F	# unsigned(15)
18 71	# unsigned(113)

FIGURE 6.6 – Définition des majeurs en CBOR

```
print (type(cbor.loads(c1)))
```

Par exemple, le programme `cbor-tag.py` retourne les résultats suivants :

```
# python3.5 cbor-tag.py
2018-05-22
c074323031382d30352d32325430303a30303a30305a
2018-05-22 00:00:00+00:00
43010203
<class 'datetime.datetime'>
```

La représentation canonique montre plus facilement le tag dans la séquence binaire :

```
C0          # tag(0)
74          # text(20)
323031382D30352D32325430303A30303A30305A # "2018-05-22T00:00:00Z"
```

Le tag 0 implique un format normalisé pour la date ; d'où l'ajout des heures, minutes et secondes, alors qu'elles n'ont pas été spécifiées initialement. On peut également remarquer que `loads` retourne un type `date` et non une chaîne de caractères.

6.7.9 Le type flottant et valeurs particulières

Le dernier type majeur (111) permet de coder les nombres flottants en utilisant la représentation définie par l'**IEEE 754**. Suivant la taille de la représentation, la suite de l'octet contient les valeurs 25 (demi précision sur 16 bits), 26 (simple précision sur 32 bits) ou 27 (double précision sur 64 bits).

Ce type permet également de coder les valeurs définies par JSON : `True` (valeur 20), `False` (valeur 21) ou `None` (valeur 22).

Finalement, ce type peut indiquer la fin d'un tableau ou d'une liste de paires quand la taille n'est pas connue au début du codage.

6.8 Questions sur CBOR

Question 6.8.1: Avantages de CBOR

Quels sont les avantages de CBOR par rapport à JSON (2 réponses) ?

- Il est plus compact dans la représentation des données.
- Il permet de représenter des nombres flottants.
- Il compresse les chaînes de caractères.
- Il est plus simple à implémenter.

Question 6.8.2: Flottant

Un flottant est-il toujours plus compact en CBOR qu'en JSON ? - Vous pouvez vous aider de <https://cbor.me>

- Oui, c'est le but de CBOR.
- Oui, pour les flottants qui ont la partie décimale à 0.
- Oui, pour les flottants de petite précision (jusqu'au centième).
- Oui, pour les flottants de grande précision (6 chiffres après la virgule).

Question 6.8.3: IgfChaîne de caractèresStrings

Soit une chaîne de caractères en CBOR.

- Elle est compressée avec un algorithme entropique (e.g. codage de Huffman).
- Elle peut contenir des caractères accentués.
- Chaque caractère est codé sur 6 bits.

Question 6.8.4: Taille variable

En CBOR, la taille d'un entier varie en fonction de sa valeur !

- Vrai
- Faux
- Ça dépend de la manière dont on a déclaré cet entier.

Question 6.8.5: Tableau

En CBOR, un tableau peut contenir des objets de types différents.

- Vrai
- Faux
- Ça dépend de la manière dont on a déclaré ce tableau.

Question 6.8.6: Fraction

On veut définir un tableau de deux éléments comme une fraction. Quel tag devra précéder la structure ? (vous pouvez vous aider du [RFC 8949](#)).

6.9 SenML

Sensor Measuring List (SenML) est une spécification qui exploite JSON ou CBOR. Elle liste un ensemble de noms/unités/mesures et les standardise en un nom de clé unique. En utilisant cette standardisation, on facilite l'interopérabilité. Les clés et valeurs sont donc réglementées et typées pour éviter tous conflits d'interopérabilité. Le format est défini dans la [RFC 8428](#) et repose sur une structure de tableau regroupant des objets comme le montre la figure suivante tirée de la RFC.

```
[{"bn" : "urn:dev:ow:10e2073a01080063", "bt":1.320067464e+09,
 "bu" : "\%RH", "v":21.2},
 {"t":10, "v":21.3},
 {"t":20, "v":21.4},
 {"t":30, "v":21.4},
 ...]
```

SenML définit les clés utilisées dans l'objet. Pour avoir une notation compacte, elles sont limitées à 1 ou 2 caractères. Parmi elles, "bn" indique un nom de base et "n" le nom d'un appareil. Si plusieurs appareils envoient la partie commune de l'identifiant de l'appareil, on peut mettre le "bn" pour éviter de le répéter à chaque fois.

Le temps de base (ou "bt") est également un moyen de compacter la notation du temps. Le temps ("t") donne le décalage et conduit à une valeur plus petite comme on le voit dans l'exemple.

L’unité de base (“bu”) indique l’unité par défaut si les autres objets ne portent pas de mot clé indiquant l’unité (“u”).

La [RFC 8428](#) définit une liste d’unités telles que le kilogramme (“kg”), le volt (“V”), etc. Dans l’exemple, “%RH” désigne un pourcentage d’humidité relative. Une valeur numérique utilise la lettre “v”, une chaîne de caractères utilise la touche “vs”.

CBOR utilise la même structure mais les petits nombres entiers positifs et négatifs sont substitués dans les clés des objets de CBOR : “bn”, “bt”, “bu” seront respectivement représentés par -1, -2 et -3 et “n”, “t”, “u” par +0, +2 et +6.

7. Mettre en œuvre un Capteur Virtuel

Les programmes relatifs à cette section se trouvent dans le répertoire plido-tp3. Il est recommandé d'utiliser la machine virtuelle avec une fenêtre de terminal pour l'objet et une autre pour le serveur.

7.1 JSON

Ca fait longtemps qu'on parle d'Internet des Objets, il est temps de mettre en pratique nos connaissances. On va commencer par une mise en oeuvre simple en Python sur votre machine. Le but dans cette partie est de tester tout ce qu'on a vu sans autre matériel qu'un ordinateur. Nous allons ouvrir deux fenêtres terminal. Dans l'une nous allons faire tourner un programme qui va émuler l'objet avec trois capteurs. Cet objet va communiquer avec un serveur qui va recueillir l'information. La communication se fera en interne avec une socket utilisant l'interface **loopback** (cf. figure 7.1 page suivante).

Youtube



7.1.1 Serveur Minimal

Listing 7.1 – minimal_server.py

```
1 import socket
2 import binascii
3
4 s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
5 s.bind(('0.0.0.0', 33033))
6
7 while True:
8     data, addr = s.recvfrom(1500)
9     print (data, ">", binascii.hexlify(data))
```

Commençons par construire un serveur sommaire (`minimal_server.py`) qui va afficher tout ce qu'il reçoit.

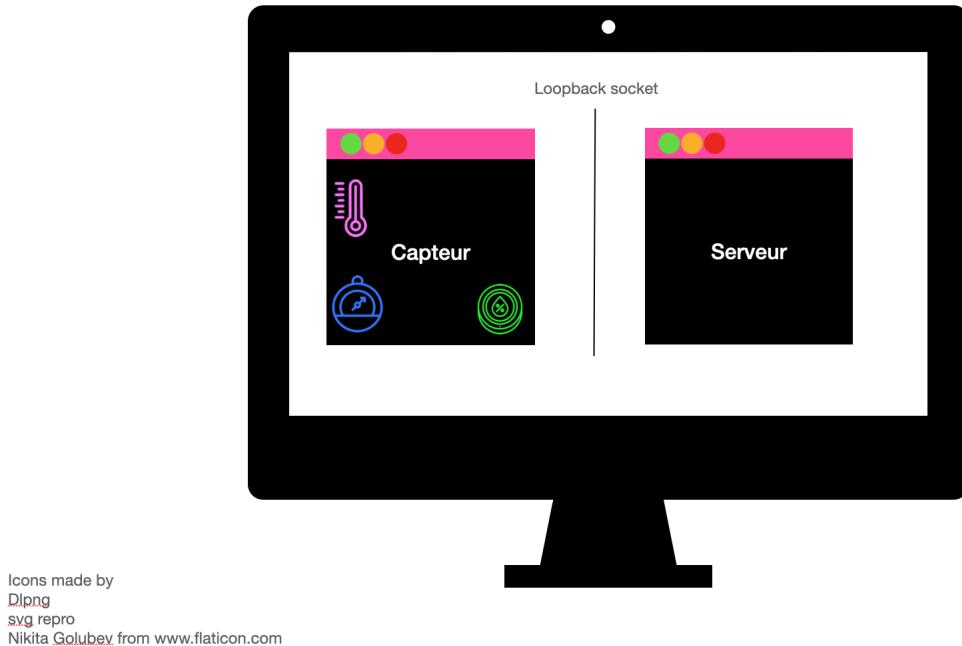


FIGURE 7.1 – Architecture client Serveur

- lignes 1 et 2 les modules `socket` pour la communication et `binascii` pour transformer le binaire en chaîne de caractères sont importés.
- ligne 4, la socket crée la socket `s` avec les paramètres pour des communications IP (`AF_INET`) et UDP (`SOCK_DGRAM`).
- ligne 5, la socket est associée au numéro de port 33033 via la fonction `bind`. L'adresse `0.0.0.0` indique que les données peuvent venir de n'importe quelle interface (Ethernet, Wi-Fi, loopback,...).
- ligne 8, dans la boucle sans fin, la fonction `recvfrom` va se bloquer dans l'attente de données. Elle retourne les données et l'adresse de l'émetteur.
- ligne 9, les données sont affichées en chaîne d'octets et en hexadécimal.

On lance le programme serveur. Comme personne lui parle, il n'affiche rien.

7.1.2 Capteur virtuel

Listing 7.2 – virtual_sensor.py

```

1 import random
2
3 class virtual_sensor:
4
5     def __init__(self, start=None, variation=None, min=None, max=None):
6         if start:
7             self.value = start
8         else:
9             self.value = 0

```

```

11         self.variation = variation
12         self.min = min
13         self.max = max
14
15     def read_value(self):
16         self.value += random.uniform(self.variation*-1, self.variation )
17
18         if self.min and self.value < self.min: self.value = self.min
19         if self.max and self.value > self.max: self.value = self.max
20
21         return self.value
22
23 if __name__ == "__main__":
24     import time
25
26     temperature = virtual_sensor(start=20, variation = 0.1)
27     pressure = virtual_sensor(start=1000, variation = 1)
28     humidity = virtual_sensor(start=30, variation = 3, min=20, max=80)
29
30     while True:
31         t = temperature.read_value()
32         p = pressure.read_value()
33         h = humidity.read_value()
34
35         print ("{:7.3f} {:10.3f} {:7.3f}".format(t, p, h))
36
37         time.sleep(1)

```

Le module **virtual_sensor** avec la classe du même nom, reflète de manière à peu près réaliste le comportement d'un capteur. On voit dans le programme principal (ligne 23 à 37) que l'on crée trois capteurs virtuels : un pour la température (ligne 31), un autre pour la pression (ligne 32), et le troisième pour l'humidité (ligne 34). L'argument `start` précise la valeur de départ, `variation` la plage de variation entre deux mesures et `min` et `max` les valeurs à ne pas dépasser. La boucle sans fin qui affiche les différentes valeurs toutes les secondes.

```

> python3 virtual_sensor.py
54.956    999.850   30.609
54.963    1000.473   32.505
55.062    1000.845   31.870
55.017    1001.619   32.257
55.083    1000.767   31.757
55.027    1001.442   31.742

```

Envoi direct

Listing 7.3 – minimal_client1.py

```

1 from virtual_sensor import virtual_sensor
2 import time
3 import socket
4
5 temperature = virtual_sensor(start=20, variation = 0.1)
6 s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
7

```

```

9  while True:
10     t = temperature.read_value()
11
12     s.sendto (t, ("127.0.0.1", 33033))
13     time.sleep(10)

```

Des scripts utilisant ce module peuvent être écrit, comme par exemple le programme `minimal_client1.py`.

La variable `t` contient la température qui est émise sur le port 33033 à l'adresse de *loopback*. On obtient ainsi une communication entre deux programmes dans votre ordinateur dont l'adresse IP locale est 127.0.0.1. Mais quand on lance le programme `minimal_client1.py`, on obtient l'erreur suivante :

```

>python3.5 minimal_client1.py
Traceback (most recent call last):
  File "minimal_client1.py", line 11, in <module>
    s.sendto (t, ("127.0.0.1", 33033))
TypeError: a bytes-like object is required, not 'float'

```

Question 7.1.1: Bug ?

Pourquoi le programme client ne fonctionne pas ?

- L'adresse IP n'est pas correcte.
- Il manque un processus de sérialisation de la donnée.
- La variable `t` n'est pas définie.
- La variable `t` ne peut pas être lue.

Envoi d'une chaîne d'octets

Listing 7.4 – `minimal_client2.py`

```

10
11     t = temperature.read_value()
12
13     s.sendto (str(t).encode(), ("127.0.0.1", 33033))

```

Dans le programme `minimal_client2.py`, le nombre flottant contenu dans la variable `t` est transformé en chaîne de caractères avec la fonction `str`, puis en chaîne d'octets avec la méthode `encode` pour être compatible avec l'argument attendu par la méthode `sendto`. Coté serveur la fonction `str` converti la chaîne d'octets reçue en flottant.

Envoi de plusieurs valeurs

Listing 7.5 – `minimal_client3.py`

```

12
13     while True:
14         t = temperature.read_value()
15         p = pressure.read_value()
16         h = humidity.read_value()
17
18         msg = "{} , {} , {}".format(t, p, h)
19         s.sendto (msg.encode(), ("127.0.0.1", 33033))

```

Pour envoyer simultanément les valeurs des trois capteurs, la représentation via une chaîne de caractères est un peu plus compliquée à mettre en œuvre. Si le programme `minimal_client3.py` utilise `format` pour envoyer des données séparées par des virgules. Côté serveur, il faut décoder cette chaîne pour y retrouver les entiers. Et ça c'est beaucoup plus complexe à faire !

JSON

Listing 7.6 – `minimal_client4.py`

```

12 while True:
13     t = temperature.read_value()
14     p = pressure.read_value()
15     h = humidity.read_value()
16
17     j = [t, p, h]
18     s.sendto (json.dumps(j).encode(), ("127.0.0.1", 33033))
19     time.sleep(10)

```

La solution la plus simple est de mettre ces trois valeurs dans un tableau python et de la transformer en une représentation JSON grâce à la fonction `dumps` du module `json`.

Cette chaîne de caractères JSON est à son tour transformée en chaîne d'octets avec `encode` et envoyée au serveur. Dans notre cas, le serveur ne fait qu'afficher la chaîne de caractères mais vous pouvez utiliser la méthode `loads` du module `json` pour déserialiser et en faire une structure Python dans le serveur, sur laquelle il est maintenant facile d'effectuer des opérations comme, par exemple, un calcul de moyenne.

```
% python3 minimal_server.py
b'[19.93044784157464, 999.1552628155773, 35.723583473834566]', => b'5b31392e39333034343738343135373436342c203939392e313535...
b'[19.940155545405723, 998.7581534530281, 35.820037116376184]', => b'5b31392e3934303135353534353430353732332c203939382e3735...
b'[20.003803212269627, 999.3517302791449, 34.33544522779677]', => b'5b32302e3030333830333231323236393632372c203939392e33353...
```

7.2 CBOR

Le passage de JSON à CBOR est très simple. Il suffit de changer un module `cbor` à la place du module de `json`. Le programme `minimal_client5.py` :

- ligne 1 fait appel au module `cbor2` en le renommant `cbor`
- le reste du programme est identique au précédent, ce n'est que dans la la sérialisation, ligne 18, que la fonction `json.dumps` est remplacée par `cbor.dumps`. Le format retourné étant une chaîne d'octets, il n'est plus nécessaire de faire appel à la méthode `encode`.

Youtube



Le programme `minimal_server.py` n'est pas modifié puisqu'il ne fait qu'afficher ce qu'il reçoit.

```
> python3 minimal_server.py
b'\x83... => b'83fb40341086f3e8b66fb408f3b7791c8d61ffb403fac15ba06088e',
b'\x83... => b'83fb40341d4d28495268fb408f33d502185c3dfb403d95a2c4981444',
```

Listing 7.7 – `minimal_client5.py`

```

1 from virtual_sensor import virtual_sensor
2 import time
3 import socket

```

```

import cbor2 as cbor
5
temperature = virtual_sensor(start=20, variation = 0.1)
7 pressure = virtual_sensor(start=1000, variation = 1)
humidity = virtual_sensor(start=30, variation = 3, min=20, max=80)
9
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
11
while True:
13     t = temperature.read_value()
14     p = pressure.read_value()
15     h = humidity.read_value()
16
17     j = [t, p, h]
18     s.sendto(cbor.dumps(j), ("127.0.0.1", 33033))
19     time.sleep(10)

```

On obtient le résultat suivant. La séquence CBOR fait 28 octets de long, l'équivalent JSON aurait fait 60 octets. Même si cela divise par deux la taille des données à transmettre, le résultat n'est pas compact. Cela tient à la représentation des nombres flottants en CBOR, car ici les flottants sont codés sur 8 octets.

Question 7.2.1: Décodage

Analysons la séquence reçue : 83fb40341086f3e8b66bfb408f3b7791c8d61ffb403fac
15ba06088e

À quoi correspond l'octet 0x83 qui commence la structure CBOR reçue ?

- au codage de l'entier positif 131.
- au codage de l'entier négatif 132.
- à la définition d'un tableau de 3 éléments.
- à la définition d'un map CBOR de 3 éléments.
- à la définition d'un tableau de taille non définie.

Question 7.2.2: Flottant

Dans cette structure, quel est le marqueur CBOR (en hexadécimal) qui indique que l'on a un nombre flottant ?

Question 7.2.3: Taille du flottant

Quelle est la taille de ce flottant en octets ?

Utilisation de nombres entiers

Pour réduire la taille des données transmises, nous allons utiliser des nombres entiers. Nous aurons besoin d'une précision au centième (deux chiffres après la virgule). Pour ce faire, il suffit, du côté du client, de prendre la partie entière du nombre multiplié par 100 et, du côté du serveur, de diviser la valeur reçue par 100. La modification du code est mineure.

Listing 7.8 – minimal_client6.py

```
j = [int(t*100), int(p*100), int(h*100)]
```

```
18     s.sendto (cbor.dumps(j), ("127.0.0.1", 33033))
```

```
> python3 minimal_server.py
b'\x83\x19\x07\xd7\x1a\x00\x01\x860\x19\x0c\xa7' => b'831907d71a0001864f190ca7',
b'\x83\x19\x07\xd4\x1a\x00\x01\x86f\x19\x0cJ' => b'831907d41a00018666190cda',
b'\x83\x19\x07\xd4\x1a\x00\x01\x86\x92\x19\rP' => b'831907d41a00018692190d50',
```

La modification est mineure et tient en 12 octets, mais il y a une diminution au niveau de l’interopérabilité, car les deux entités doivent connaître la transformation de la valeur liée à la multiplication par 100.

Question 7.2.4: Anticipons la tailleAnticipating the size

Quelle est la taille minimale et maximale de la structure CBOR envoyée, en prenant en compte les valeurs possibles.



8. Séries temporelles

Les capteurs virtuels que nous avons programmés jusqu'à présent émettent les données à chaque fois que celles-ci sont mesurées. Cela permet au serveur de suivre en temps réel le comportement du système étudié. Mais dans certains cas, le temps réel n'est pas nécessaire et il est préférable de limiter le nombre d'émissions, par exemple pour économiser l'énergie du capteur.

Pour ce faire, nous pouvons utiliser un tableau qui va accumuler les valeurs et l'envoyer quand le tableau atteint une certaine taille.

Youtube



8.1 Envoi d'un tableau

Le programme `minimal_humidity1.py` accumule les données dans un tableau `h_humidity` quand celui-ci atteint 30 éléments (ligne 17), les données sont envoyées au serveur.

Listing 8.1 – `minimal_humidity1.py`

```
from virtual_sensor import virtual_sensor
import time
import socket
import cbor2 as cbor

humidity      = virtual_sensor(start=30, variation = 3, min=20, max=80)

s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
NB_ELEMENT = 30
h_history = []

while True:
    h = int(humidity.read_value())*100

    # No more room to store value, send it.
    if len(h_history) == NB_ELEMENT:
```

```

18     if len(h_history) >= NB_ELEMENT:
19         s.sendto (cbor.dumps(h_history), ("127.0.0.1", 33033))
20         h_history = []
21
22     h_history.append(h)
23     print (len(h_history), len(cbor.dumps(h_history)), h_history)
24
25     time.sleep(10)

```

```

1 4 [3241]
2 7 [3241, 2945]
3 10 [3241, 2945, 2762]
4 13 [3241, 2945, 2762, 2625]
5 16 [3241, 2945, 2762, 2625, 2480]
6 19 [3241, 2945, 2762, 2625, 2480, 2769]

```

Le premier chiffre de la ligne indique le nombre d'éléments et le second la taille dans le codage CBOR. On remarque que l'ajout d'un élément augmente la taille du tableau de 3 octets. Les valeurs correspondant à une mesure d'humidité ne varient pas fortement. Ainsi un tableau de 30 mesures a une taille de 92 octets.

8.2 Codage par différence

On peut optimiser le volume de données transférées en utilisant un codage par delta (i.e. la variation de l'humidité). La première valeur du tableau correspond à la valeur mesurée tandis que les suivantes représentent la différence entre la valeur mesurée et la précédente.

Listing 8.2 – minimal_humidity2.py

```

18     if len(h_history) == 0:
19         h_history = [h]
20     elif len(h_history) >= NB_ELEMENT:
21         print ("send")
22         s.sendto (cbor.dumps(h_history), ("127.0.0.1", 33033))
23         h_history = [h]
24     else:
25         h_history.append(h-prev)
26
27     prev = h

```

Le programme `minimal_humidity2.py` gère différemment le remplissage du tableau :

- ligne 14 et 15, si le tableau est vide, le tableau est créé avec la valeur mesurée,
- ligne 16 à 22, sinon si le tableau est plein, il est sérialisé en CBOR et envoyé au serveur, puis réinitialisé avec la valeur mesurée,
- ligne 23 et 24, sinon la différence entre la précédente valeur et celle mesurée est stockée dans le tableau.

Le listing suivant donne un exemple d'exécution.

```

1 4 [2521]
2 6 [2521, 79]
3 8 [2521, 79, 224]
4 10 [2521, 79, 224, -40]

```

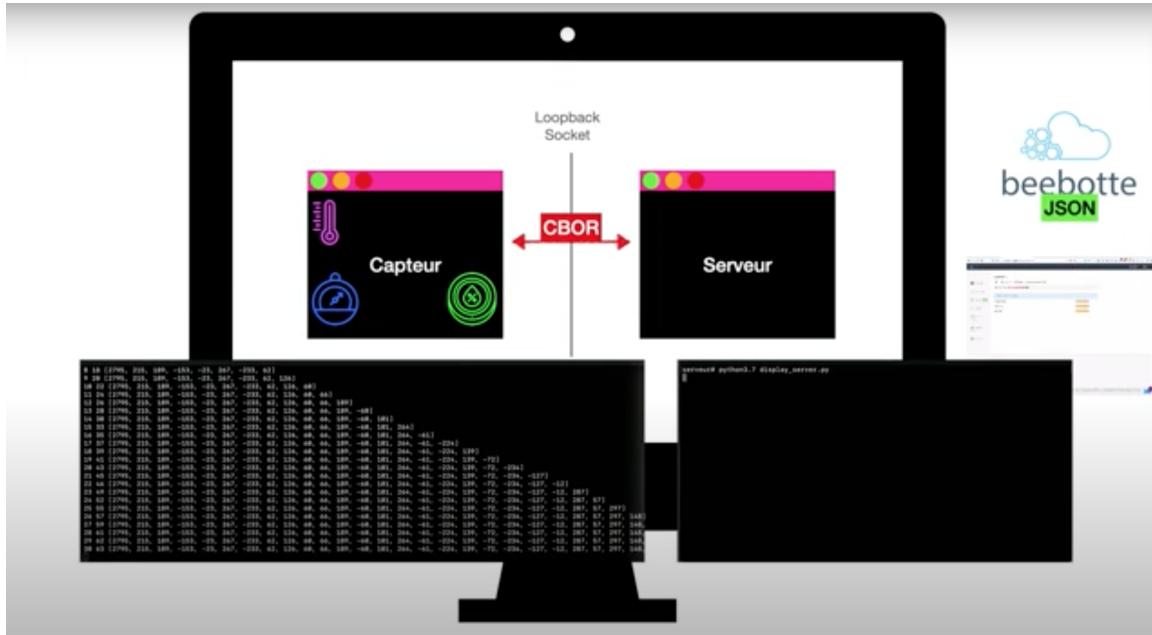


FIGURE 8.1 – Architecture Client/Serveur

```

5 12 [2521, 79, 224, -40, -112]
6 13 [2521, 79, 224, -40, -112, 1]
7 15 [2521, 79, 224, -40, -112, 1, 130]
8 18 [2521, 79, 224, -40, -112, 1, 130, -288]
9 21 [2521, 79, 224, -40, -112, 1, 130, -288, 299]

```

Ceci met en valeur deux souplesses de CBOR :

- la taille du tableau est dynamique. Si l'on change le nombre de valeurs à transmettre, le tableau l'indique et l'on n'a pas besoin de modifier le code du récepteur ;
- la taille des données dépend de leur valeur. Pour les variations entre -24 et +23, un seul octet sera nécessaire. On le voit sur l'exemple précédent : l'ajout de la valeur '1' dans le tableau fait passer la taille de la représentation CBOR de 12 à 135 octets. Les valeurs entre 256 et +255 sont transmises sur 2 octets ; il est donc possible de cette manière d'optimiser la transmission sans ajouter de contrainte. S'il y avait une brusque variation de l'humidité, la représentation CBOR s'adapterait pour la transmettre.

La taille est réduite d'un tiers (environ 66 octets) pour transmettre la même information.

8.3 Architecture

La figure 8.1 représente l'architecture générale du système. Le programme `minimal_humidity2.py` fournit les séries temporelles. Il reste à définir le programme serveur qui va les traiter et faire appel à un autre service pour les afficher sous forme de graphe.

Si l'on suit le flux d'information, le capteur va produire des données au format CBOR pour être compact et le programme serveur va transformer cette information en une structure JSON respectant les spécifications du service d'affichage.

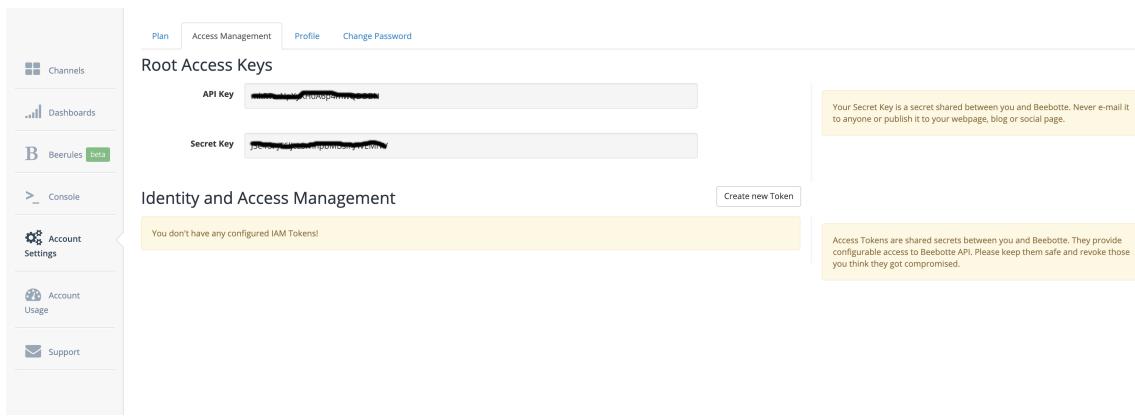


FIGURE 8.2 – Clé et secret pour l’authentification

8.4 Beebotte

Il existe plusieurs sites qui permettent de le faire. Nous allons utiliser <https://beebotte.com>, mais ce que nous allons présenter peut très bien s’appliquer à d’autres sites.

8.4.1 Configuration

La première étape consiste à créer un compte en cliquant sur *Sign Up* sur la page de garde et en remplissant un formulaire classique avec votre login, adresse de courrier électronique et mot de passe. Une fois le compte validé, le service est accessible.

Le compte nous permet de nous authentifier pour gérer les données sur le site, mais il faut également disposer d’autorisation pour pouvoir y déposer des données via l'**API REST**. Pour cela, il faut se rendre sur la page *Account Setting* puis l’onglet *Access Management*. Cette page (cf. figure 8.2) donne une clé et un secret pour gérer l’ensemble des données sur le site.

Notez ces valeurs et stockez les dans un fichier `config_bbt.py` qui a cet aspect (vos valeurs sont forcément différentes) :

Listing 8.3 – config_bbt.py

```
1 API_KEY      = "GAJ3SFmUZSXmB2zqdcmczuXc"
2 SECRET_KEY   = "4NCsrM1cfmFdMZF4E47aTfmCaU3UfyQo"
```



Nous allons maintenant créer un canal (*channel*) dans lequel nous allons définir les objets correspondant aux capteurs. En Cliquant sur *Channels* puis *Create New*, la page représentée figure ?? page ?? apparaît.

Il faut donner un nom au channel (*capteurs* dans l'exemple), cocher la case *public* et créer trois ressources pour les trois valeurs qui nous intéressent (*temperature*, *humidity*, *presure*) et faire correspondre les unités.

8.4.2 Enregistrement des ressources

Le programme `display_server.py` permet de correspondre avec Beebotte via son API REST. Il commence par l’importation des modules nécessaires :

Listing 8.4 – display_server.py

```

1 import socket
2 import binascii
3 import cbor2 as cbor
4 import beebotte
5 import config_bbt #secret keys
6 import datetime
7 import time
8 import pprint

```

- ligne 4, le module Python beebotte est disponible pour simplifier la manipulation des données¹.
- ligne 5, le module contient la clé et le secret nécessaire à la connexion obtenu précédemment.

```

10 s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
11 s.bind((‘0.0.0.0’, 33033))

```

- ligne 10 et 11 permettent d’ouvrir la socket pour communiquer avec les capteurs.

```

12 bbt = beebotte.BBT(config_bbt.API_KEY, config_bbt.SECRET_KEY)

```

- ligne 13 une instance permettant la connexion avec les serveurs de Beebotte est définie grâce à la fonction BBT. Les paramètres de connexion provenant du module config_bbt sont pris en compte.

```

while True:
14     data, addr = s.recvfrom(1500)

16     j = cbor.loads(data)
17     to_bbt("capteurs", "temperature", j, factor=0.01)

```

Dans le programme principal, un boucle sans fin attend la série temporelle codée en CBOR venant du capteur (ligne 42), les transforme tableau Python (ligne 44) et appelle la fonction to_btt en précisant :

- le canal et la ressource qui ont été définie précédemment sur Beebotte ;
- la série temporelle ;
- la précision pour transformer ces entiers en flottant.

```

def to_bbt(channel, res_name, cbor_msg, factor=1, period=10, epoch=None):
16     global bbt

18     prev_value = 0
19     data_list = []
20     if epoch:
21         back_time = epoch
22     else:
23         back_time = time.mktime(datetime.datetime.now().timetuple())
24
25     back_time -= len(cbor_msg)*period
26

```

1. S'il n'était pas présent sur votre ordinateur, vous devriez l'installer avec la commande pip3 install beebotte.

```

28     for e in cbor_msg:
29         prev_value += e
30
31         back_time += period
32
33         data_list.append({"resource": res_name,
34                           "data" : prev_value*factor,
35                           "ts" : back_time*1000} )
36
37         pprint.pprint (data_list)
38
39         bbt.writeBulk(channel, data_list)

```

La fonction `to_bbt` fait l'essentiel du travail de transformation. Elle prend en argument :

- le nom du canal créé sur Beebotte. Dans notre cas, ce sera `capteurs`;
- le nom de l'objet dans ce canal que nous avons également créé sur le site web. Dans notre cas, ce sera `humidity`;
- le tableau Python des mesures codées en delta;
- le facteur multiplicatif, c'est-à-dire la précision. Ici, il faudra diviser par 100;
- la période entre deux mesures; cela nous permettra de calculer l'instant de la mesure. Par défaut, la période est de 10 secondes;
- le temps de réception du message pour dater les échantillons. S'il n'est pas spécifié, le temps courant est pris.

Cette fonction transforme le tableau Python suivant :

```
[3311, 124, -144, -188, -94, 289, -1, -72, 1 ...]
```

en un tableau de dictionnaire :

```
[{'data': 33.11, 'resource': 'humidity', 'ts': 1596730115000.0},
 {'data': 34.35, 'resource': 'humidity', 'ts': 1596730125000.0},
 {'data': 32.91, 'resource': 'humidity', 'ts': 1596730135000.0},
 {'data': 31.03, 'resource': 'humidity', 'ts': 1596730145000.0},
 ...]
```

Chaque dictionnaire contient trois éléments imposés par Beebotte :

- le nom de la ressource (`resource`) telle qu'elle a été définie sur l'interface pour le canal;
- la valeur associée pour cette ressource (`data`);
- l'instant à laquelle cette mesure a été faite (`ts`). Le temps est représenté suivant le format **Epoch** qui compte le nombre de secondes depuis le premier Janvier 1970².

Le calcul du *timestamp* (`ts`) est l'opération la plus complexe de cette fonction mais les module `time` et `datetime` facilitent le calcul. Si l'argument `epoch` a été fourni lors de l'appel, la fonction prend cette valeur, sinon le calcule ligne 23. La fonction `now` retourne la date et l'heure courante, qui est transformé en un tuple grâce à la fonction `timetuple`. A partir de ce dernier, la fonction `maketime` le converti en epoch.

Ligne 25, l'epoch à laquelle la première mesure du tableau a été faite est calculé en prenant le temps actuel (cela suppose que l'on néglige le temps de traitement et de transmission) auquel on

2. voir <https://www.epochconverter.com/> pour les conversions.

Configured resources		
temperature	No Persisted Data	
pressure	No Persisted Data	
humidity	26.51 %	2 minutes ago

FIGURE 8.3 – État des ressources

FIGURE 8.4 – Crédit d'un widget

retranche la durée de la capture, c'est-à-dire comme le nombre d'éléments du tableau multiplié par l'intervalle entre chaque mesure (*period*).

Ligne 32 à 34 la structure attendue par Beebotte est construite. le résultat est envoyé, ligne 38, grâce à la fonction `writeBulk` qui permet d'envoyer un ensemble de valeurs dans un tableau.

On peut vérifier que Beebotte a reçu des données en visualisant le canal capteurs sur l'interface Web. On peut voir sur la figure 8.3 que seule la ressource `humidity` a reçu des données. L'interface affiche la dernière valeur reçue et la date de réception.

8.4.3 Visualisation des ressources

Maintenant que les ressources sont stockées dans les serveurs de Beebotte, il est possible de les visualiser graphiquement, en allant dans *Dashboard* puis *create Dashboard* et *Add Widget* pour sélectionnez un widget comme *Multi-line chart*.

Puis, configurez le widget en définissant le canal et la ressource de ce canal comme le montre la figure 8.4.

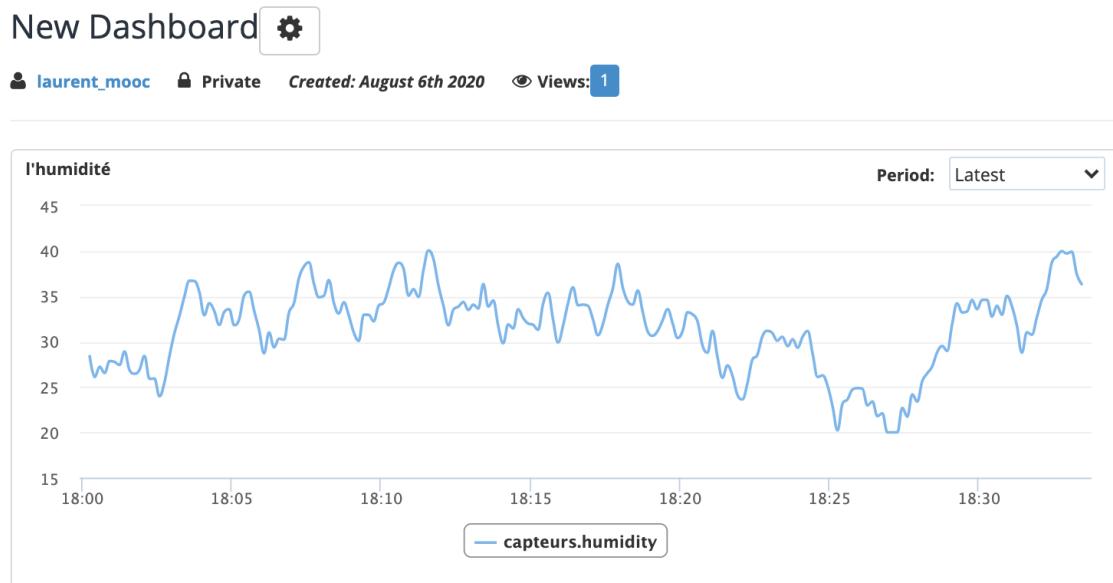


FIGURE 8.5 – Suivi de l’humidité

En retournant sur le dashboard, on peut voir l’évolution de l’humidité au cours du temps (cf. figure 8.5).

8.5 Interopérabilité

La chaîne de collecte de l’information que nous venons de construire allant du capteur à l’affichage, n’est pas complètement interopérable. Certes le capteur envoie des données au format CBOR qui peuvent être interprété par l’autre extrémité, mais le récepteur ne sait pas :

- qu’il s’agit d’une série temporelle codée avec des deltas ; that it is a coded time series with deltas ;
- que les données ont été multipliée par 100 pour pouvoir envoyer des nombres entiers, plus compacts sans perdre trop de précision ; that the data have been multiplied by 100 to be able to send whole numbers, more compact without losing too much precision ;
- que le pas de mesure est de 10 secondes ; that the measurement step is 10 seconds ;
- que les données concernent le taux d’humidité. that the data are related to the moisture content.

Ces informations ont été précisées dans le programme `display_server.py`, de même la transformation de la structure de tableau de la série temporelle en un dictionnaire avec des mots clés spécifique à Beebotte on été gravé dans le programme.

Nous verrons par la suite comment améliorer cette interopérabilité.

8.6 et SenML ?

Dans la communication avec Beebotte, le site structure l’envoi des mesures en définissant un dictionnaire JSON avec des mots clés particuliers. Pour utiliser un autre site, le format des échanges doit être modifié même si les informations restent identiques.

De plus, lors de la configuration des ressources sur le site de Beebotte, la nature de la mesure a du être précisée ; par exemple, s'il s'agit d'une température, d'un taux d'humidité... Il faut également parfois indiquer le type de la mesure (texte, entier, flottant...) voire les unités.

SenML défini dans le [RFC 8428](#) propose une structuration des données fournie par le capteur. Pour réduire l'impact de la transmission, les noms des champs ont été choisis pour être le plus compact possible. Par exemple, la lettre v va indiquer une valeur (à comparer avec la clé data utilisée lors de la communication avec Beebotte). Pour être encore plus compact, la représentation en CBOR utilisera des entiers courts au lieu de caractères.

Il est également possible de transporter l'unité de la mesure avec le mot clé u .

SenML ne définit pas que des unités du système international, mais également des unités secondaires pour limiter la taille de la représentation. Il sera plus compact de transmettre :

```
{"u": "MHz", "v": 868}
```

que

```
{"u": Hz", "v": 868000000}.
```

Le standard définit aussi des temps de base et des valeurs de base auxquelles les temps et les valeurs vont se référer ; ce qui permet également de réduire la taille des valeurs. Finalement, le ou les objets peuvent s'identifier dans les données transmises en définissant un nom de base (bn : *base name*), le nom du capteur (n : *name*) vient compléter le nom de base.

Émission

Listing 8.5 – minimal_senml_client.py

```
1 from virtual_sensor import virtual_sensor
2 import time
3 import socket
4 import json
5 import kpn_senml as senml
6 import pprint
7 import binascii
8 import datetime
9 import time
10 import pprint
11
12
13 NB_ELEMENT = 5
14
15
16 temperature = virtual_sensor(start=20, variation = 0.1)
17 pressure = virtual_sensor(start=1000, variation = 1)
18 humidity = virtual_sensor(start=30, variation = 3, min=20, max=80)
19
20
21 s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
22
23 while True:
24     pack = senml.SenmlPack("device1")
25     pack.base_time = time.mktime( datetime.datetime.now().timetuple() )
26
27     for k in range(NB_ELEMENT):
28         if k == 0:
29             pack.add("temperature", "v", 20)
30             pack.add("temperature", "u", "MHz")
31             pack.add("pressure", "v", 1000)
32             pack.add("pressure", "u", "Hz")
33             pack.add("humidity", "v", 30)
34             pack.add("humidity", "u", "MHz")
35
36         else:
37             pack.add("temperature", "v", temperature.get())
38             pack.add("pressure", "v", pressure.get())
39             pack.add("humidity", "v", humidity.get())
40
41         s.sendto(pack.to_json(), ("127.0.0.1", 12345))
```

```

27     t = round(temperature.read_value(), 2)
28     h = round(humidity.read_value(), 2)
29     p = int(pressure.read_value() *100) # unit is Pa not hPa
30
31
32     rec = senml.SenmlRecord("temperature",
33         unit=senml.SenmlUnits.SENML_UNIT_DEGREES_CELSIUS,
34         value=t)
35     rec.time = time.mktime(datetime.datetime.now().timetuple())
36     pack.add(rec)
37
38     rec = senml.SenmlRecord("humidity",
39         unit=senml.SenmlUnits.SENML_UNIT_RELATIVE_HUMIDITY,
40         value=h)
41     rec.time = time.mktime(datetime.datetime.now().timetuple())
42     pack.add(rec)
43
44     rec = senml.SenmlRecord("pressure",
45         unit=senml.SenmlUnits.SENML_UNIT_PASCAL,
46         value=p)
47     rec.time = time.mktime(datetime.datetime.now().timetuple())
48     pack.add(rec)
49
50     time.sleep(10)
51
52     pprint.pprint(json.loads(pack.to_json()))
53     print ("JSON length:", len(pack.to_json()), "bytes")
54     print ("CBOR length:", len(pack.to_cbor()), "bytes")
55
56     s.sendto(pack.to_cbor(), ("127.0.0.1", 33033))

```

Le programme `minimal_senml_client.py` illustre le fonctionnement de SenML. Il repose sur deux objets :

- l’objet `SenmlPack` inclus les informations communes à l’objet, comme le nom de base (ici `device1` ligne 23) ou la base de temps, ligne 24.
- l’objet `SenmlRecord` contient une mesure où l’on peut préciser son nom, son unité et sa valeur (lignes 32, 38 et 44). Le temps est également précisé lignes 35, 41 et 47. Ces enregistrements sont ajoutés à l’objet `pack`.

Le programme récupère les trois valeurs de température, humidité et pression (lignes 27 à 29) en les arrondissant à 2 chiffres après la virgule pour la température et l’humidité et convertit la pression, d’hecto Pascal en Pascal puisque c’est l’unité définie par SenML.

Les mesures se font toutes les 10 secondes (délais ligne 48) et quand le nombre de mesures défini ligne 13 est atteint, le codage SenML en CBOR est envoyé au serveur.

```

[{'bn': 'device1',
 'bt': 1650463643.0,
 'n': 'temperature',
 't': 0.0,
 'u': 'Cel',
 'v': 20.08},
 {'n': 'humidity', 't': 0.0, 'u': '%RH', 'v': 31.1},
 {'n': 'pressure', 't': 0.0, 'u': 'Pa', 'v': 99920}]
JSON length: 197 bytes

```

```
CBOR length: 126 bytes
```

Ce premier listing montre le premier enregistrement pour les trois grandeurs mesurées. Il s'agit d'un tableau de 3 éléments. Le premier contient les valeurs de bases (ici le nom et l'heure de référence) suivie de la grandeur à mesurer, de son unité et sa valeur. Le deuxième et le troisième éléments, mettent à jours le nom de la grandeur, son unité et sa valeur, les autres informations précédemment définies restent valables.

```
[{'bn': 'device1',
 'bt': 1650463643.0,
 'n': 'temperature',
 't': 0.0,
 'u': 'Cel',
 'v': 20.08},
 {'n': 'humidity', 't': 0.0, 'u': '%RH', 'v': 31.1},
 {'n': 'pressure', 't': 0.0, 'u': 'Pa', 'v': 99920},
 {'n': 'temperature', 't': 10.0, 'u': 'Cel', 'v': 20.04},
 {'n': 'humidity', 't': 10.0, 'u': '%RH', 'v': 31.49},
 {'n': 'pressure', 't': 10.0, 'u': 'Pa', 'v': 99872}]
JSON length: 361 bytes
CBOR length: 232 bytes
```

Quand on ajoute 10 secondes plus tard de nouvelles mesures, un temps relatif de 10 secondes est indiqué pour l'enregistrement des températures et il reste valable pour les enregistrements suivants.

Question 8.6.1: Codage

A quoi correspond la clé { 'u' : 'Cel' } que l'on retrouve dans la structure précédente ?

Question 8.6.2: Accroissement

Dans les deux représentations JSON et CBOR, de combien la taille est-elle accrue par l'ajout des mesures effectuées ? d'où viennent ces différences ?

Question 8.6.3: Une seule grandeur

Si on ne s'intéressait qu'à une seule grandeur, par exemple l'humidité. A quoi ressemblerait la structure SenML en JSON ?

Réception

Le traitement par le module SenML tel qu'il est mis en œuvre n'est pas complet, il ne gère pas correctement les timestamps. Mais, il n'est pas vraiment nécessaire pour traiter ces messages. En effet, comme on l'a vu précédemment, les valeurs SenML sont composés d'une base et d'une valeur. En concaténant les différents éléments du tableau, on garde les clés de base et on remplace les clés qui se répètent à chaque éléments. Cela permet d'avoir une mise en œuvre très simple du décodage, pour la série SenML qui nous est fournie.

Listing 8.6 – minimal_senml_server.py

```
import socket
import pprint
```

```

3 import binascii
4 import pprint
5 import cbor2 as cbor
6
7 import beebotte
8 import config_bbt #secret keys
9
10 naming_map = {'bn': -2, 'bt': -3, 'bu': -4, 'bv': -5, 'bs': -16,
11     'n': 0, 'u': 1, 'v': 2, 'vs': 3, 'vb': 4,
12     'vd': 8, 's': 5, 't': 6, 'ut': 7}
13
14 s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
15 s.bind((‘0.0.0.0’, 33033))
16
17 bbt = beebotte.BBT(config_bbt.API_KEY, config_bbt.SECRET_KEY)
18
19 while True:
20     data, addr = s.recvfrom(1500)
21
22     sml_data = cbor.loads(data)
23
24     sml_record = {}
25     bbt_record = []
26
27     for e in sml_data:
28         sml_record = {**sml_record, **e} # merge dict
29         print (sml_record)
30
31         ts = sml_record[naming_map["t"]]
32         if naming_map["bt"] in sml_record:
33             ts += sml_record[naming_map["bt"]]
34
35         res = sml_record[naming_map["n"]]
36
37         data = sml_record[naming_map["v"]]
38         if naming_map["bv"] in sml_record:
39             data += sml_record[naming_map["bv"]]
40
41
42         bbt_record.append({"resource": res, "data": data, "ts": ts*1000})
43
44     pprint.pprint (bbt_record)
45     channel = sml_record[naming_map["bn"]]
46     bbt.writeBulk(channel, bbt_record)
47

```

Le programme `minimal_senml_server.py` va convertir le format SenML codé en CBOR dans le format attendu par Beebotte. La version CBOR utilise des nombres plutôt que des tags. Le dictionnaire `naming_map` défini lignes 10 à 12 permet la correspondance utilisée par la suite pour rendre le code plus lisible.

Les lignes 14 à 17 initialisent les communications venant du capteur et celles allant à Beebotte.

Les données reçues ligne 21 sont transformées en structure Python ligne 23. Cette correspondance est possible car Python autorise des clés numériques et celles-ci ne sont pas répétées plusieurs fois dans une map CBOR.

La boucle commençant ligne 28 permet d'explorer tous les éléments du tableau SenML, les nouvelles entrées sont fusionnées avec les anciennes (ligne 29)³.

Les informations concernant le temps sont ensuite recherchées. D'abord le temps (ligne 32) et s'il un temps de base existe (ligne 33) il est ajouté. On procède de même pour la valeur (lignes 38 à 40). Pour le nom, il n'y a pas de concaténation car le nom de base sera utilisé comme canal Beebotte, il est récupéré à la fin ligne 46.

A partir de ces informations, la structure attendue par Beebotte est construite ligne 43 en ajoutant le dictionnaire dans le tableau `bbt_record`.

Ligne 47, l'information est envoyée à Beebotte. Si les clés d'authentification, le nom du canal et des ressources sont correct, les information s'affiche sur le site, comme précédemment.

```
{0: {'temperature': 2: 19.94, 6: 0.0, 1: 'Cel', -2: 'device1', -3: 1650463732.0}
{0: {'humidity': 2: 27.7, 6: 0.0, 1: '%RH', -2: 'device1', -3: 1650463732.0}
{0: {'pressure': 2: 100109, 6: 0.0, 1: 'Pa', -2: 'device1', -3: 1650463732.0}
{0: {'temperature': 2: 19.88, 6: 10.0, 1: 'Cel', -2: 'device1', -3: 1650463732.0}
{0: {'humidity': 2: 24.82, 6: 10.0, 1: '%RH', -2: 'device1', -3: 1650463732.0}
{0: {'pressure': 2: 100056, 6: 10.0, 1: 'Pa', -2: 'device1', -3: 1650463732.0}
{0: {'temperature': 2: 19.93, 6: 20.0, 1: 'Cel', -2: 'device1', -3: 1650463732.0}
{0: {'humidity': 2: 23.74, 6: 20.0, 1: '%RH', -2: 'device1', -3: 1650463732.0}
{0: {'pressure': 2: 100123, 6: 20.0, 1: 'Pa', -2: 'device1', -3: 1650463732.0}
{0: {'temperature': 2: 19.92, 6: 30.0, 1: 'Cel', -2: 'device1', -3: 1650463732.0}
{0: {'humidity': 2: 25.82, 6: 30.0, 1: '%RH', -2: 'device1', -3: 1650463732.0}
{0: {'pressure': 2: 100220, 6: 30.0, 1: 'Pa', -2: 'device1', -3: 1650463732.0}
{0: {'temperature': 2: 19.9, 6: 40.0, 1: 'Cel', -2: 'device1', -3: 1650463732.0}
{0: {'humidity': 2: 24.47, 6: 40.0, 1: '%RH', -2: 'device1', -3: 1650463732.0}
{0: {'pressure': 2: 100173, 6: 40.0, 1: 'Pa', -2: 'device1', -3: 1650463732.0}
[{'data': 19.94, 'resource': 'temperature', 'ts': 1650463732000.0},
{'data': 27.7, 'resource': 'humidity', 'ts': 1650463732000.0},
{'data': 100109, 'resource': 'pressure', 'ts': 1650463732000.0},
{'data': 19.88, 'resource': 'temperature', 'ts': 1650463742000.0},
{'data': 24.82, 'resource': 'humidity', 'ts': 1650463742000.0},
{'data': 100056, 'resource': 'pressure', 'ts': 1650463742000.0},
{'data': 19.93, 'resource': 'temperature', 'ts': 1650463752000.0},
{'data': 23.74, 'resource': 'humidity', 'ts': 1650463752000.0},
{'data': 100123, 'resource': 'pressure', 'ts': 1650463752000.0},
{'data': 19.92, 'resource': 'temperature', 'ts': 1650463762000.0},
{'data': 25.82, 'resource': 'humidity', 'ts': 1650463762000.0},
{'data': 100220, 'resource': 'pressure', 'ts': 1650463762000.0},
{'data': 19.9, 'resource': 'temperature', 'ts': 1650463772000.0},
{'data': 24.47, 'resource': 'humidity', 'ts': 1650463772000.0},
{'data': 100173, 'resource': 'pressure', 'ts': 1650463772000.0}]
```

Le listing précédent montre cette transformation. Les premières lignes correspondent aux enregistrements fusionnées et le tableau final, ce qui a été envoyé à Beebotte.

Question 8.6.4: base value

Pourrait-on utiliser le champ SenML *base value* pour diminuer la taille des données de pression atmosphérique ?

3. Dans les version plus récentes de Python, il est possible d'utiliser l'opérateur `|`.

9. Découvrons le LoPy

Les programmes relatifs à cette section se trouvent dans le répertoire `plido-tp3` pour le serveur et `pycom` pour le LoPy.

9.1 Introduction

Grâce aux émulateurs de capteurs décrits au chapitre précédent, vous avez pu appliquer les concepts essentiels de l’IoT sur votre ordinateur.

Cependant, si vous le pouvez, nous vous invitons à le faire sur de vrais objets connectés en utilisant des **LoPy4** (plateforme de prototypage IoT) de la société **Pycom** et des capteurs de température, humidité et pression **BME280** (cf. figure 9.1).

Un LoPY4 se programme en Python (ou plutôt **micro-python** qui est la version du langage pour systèmes embarqués) pour traiter les données. Dans un premier temps, nous allons utiliser le Wi-Fi pour communiquer avec votre ordinateur mais, par la suite, nous mettrons en place une

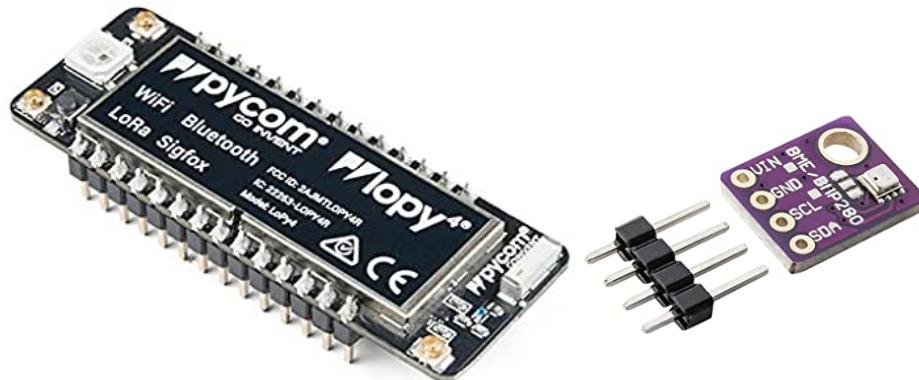


FIGURE 9.1 – LoPY4 et capteur BME280

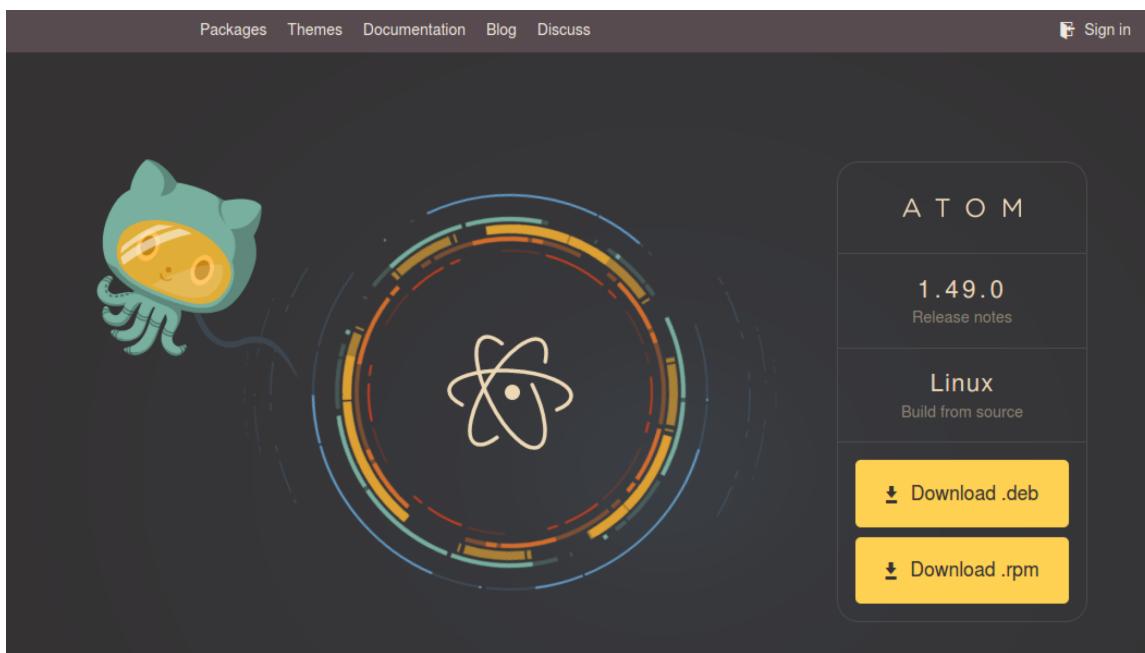


FIGURE 9.2 – Page d'accueil d'Atom

communication via **LoRaWAN** ou **Sigfox** qui peut vous demander plus de configuration mais vous permettra de mieux comprendre ces protocoles.

Même si vous n'avez pas de LoPy4, vous pouvez parcourir cette section pour voir les contraintes supplémentaires liées aux objets connectés.

9.2 Installation d'Atom

Atom est un éditeur de texte performant, spécifiquement conçu pour le codage en différents langages. Atom va nous aider à programmer en Python et va également gérer la communication avec notre LoPy via le port **USB** (grâce au plugin **pymakr**). Atom fonctionne à peu près de la même manière sur **Mac OS**, **Windows** et **Linux**, mais en s'adaptant aux particularités du système d'exploitation (place dans les menus, nom des liens séries...). Donc, il se peut que vous ayez quelques différences entre ce que vous avez dans cet ouvrage et l'écran de votre ordinateur. Les menus et sites Web indiqués peuvent également changer au cours du temps même si nous nous efforçons de faire des mises à jour régulières du cours.

Pour commencer à programmer avec votre LoPy, vous devez installer sur votre ordinateur le logiciel Atom (voir sur <http://atom.io>). Vous pouvez télécharger le package correspondant à votre système d'exploitation (cf. figure suivante).

— Pour Mac et Windows, cliquez sur l'icône "télécharger" pour l'installer¹.

1. il y a des risques d'incompatibilité des dernières versions d'Atom avec le package de gestion du LoPy (pymakr). Nous vous recommandons d'utiliser une version plus ancienne, comme la 1.43 disponible dans les archives d'Atom Release <https://github.com/atom/atom/releases/tag/v1.43.0>(AtomSetup-x64.exe pour Windows ou atom-mac.zip pour Mac OS).

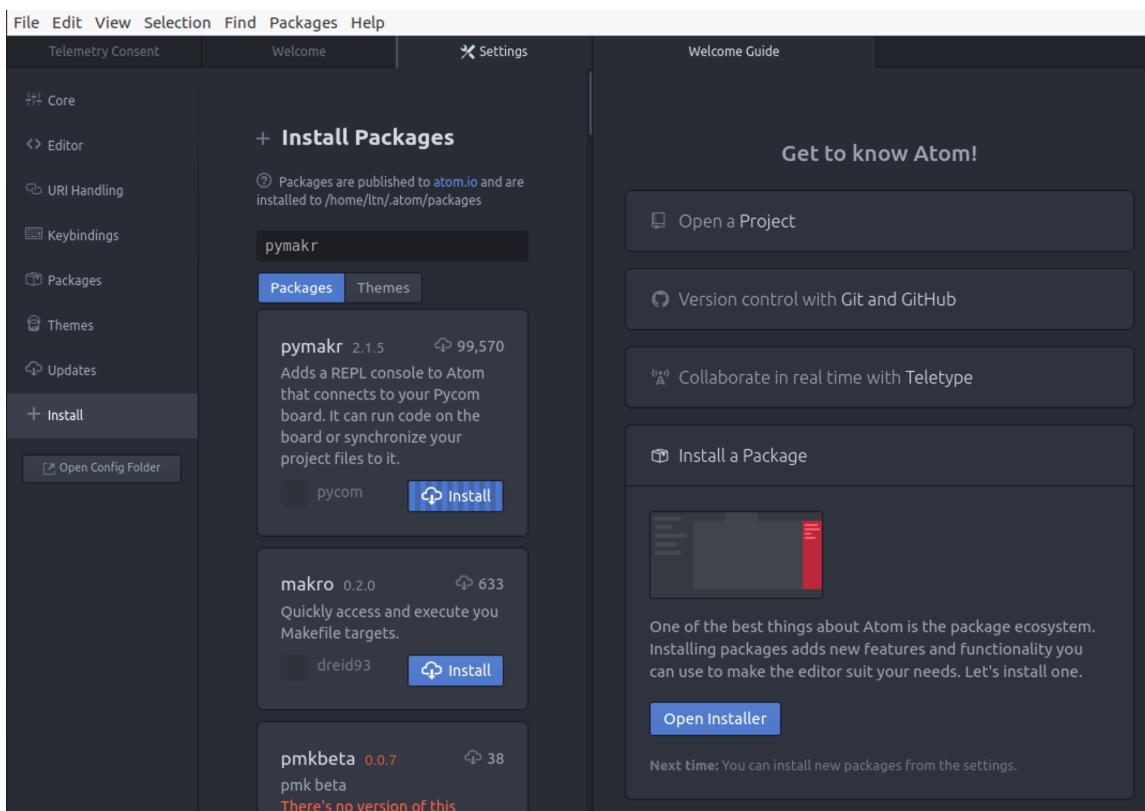


FIGURE 9.3 – Installation de Paquetages

— Pour Linux, téléchargez le .deb et tapez `sudo dpkg -i atom-amd64.deb`². Lancez Atom en cliquant sur l’icône ou, sous Linux, en tapant `atom` dans un terminal.

Lancez Atom en cliquant sur l’icône ou, sous Linux, en tapant `atom` dans un terminal. L’écran d’accueil apparaît.

9.2.1 Communiquez avec votre Pycom

Pour communiquer avec le Pycom à travers Atom, vous devez installer le package `pymakr`.

Cliquez sur *Install a Package* puis *Open Installer*. Une autre fenêtre s’ouvre (cf. figure 9.3). Tapez `pymakr` dans le menu. Un package apparaît portant ce nom. Cliquez sur *Install*. L’installation peut prendre plusieurs minutes. Vous avez le temps de prendre un café.

Une fois le café bu et l’installation terminée, une nouvelle fenêtre (terminal) s’ouvre en bas d’Atom.

Ce terminal (cf. figure 9.4 page ci-contre) vous permettra de dialoguer avec le LoPy. Branchez le LoPy à votre ordinateur. Vous devriez voir l’invite `>>>` caractéristique d’un interpréteur Python³.

Toutes les commandes que vous allez taper dans cette fenêtre vont s’exécuter sur votre LoPy. Par

2. Il est possible qu’un message vous dise que git n’est pas installé. Dans ce cas, tapez `sudo apt-get install`

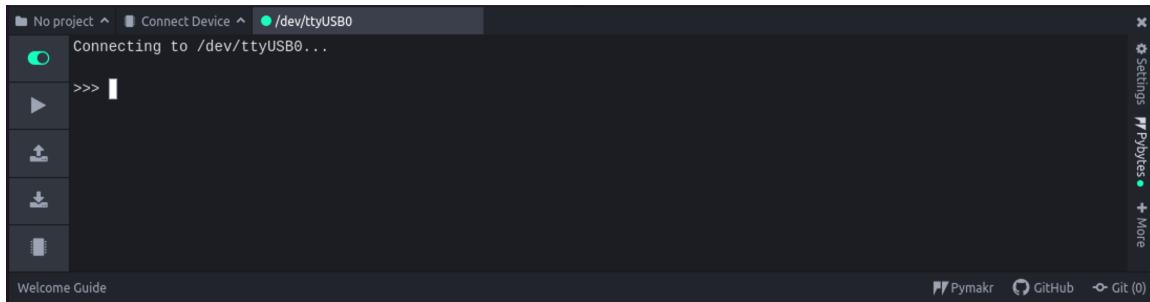


FIGURE 9.4 – Fenêtre Pymakr

exemple, si vous tapez⁴ :

```
Connecting to /dev/ttyUSB0...
>>> 1+1
2
>>>
```

l'addition se fait sur le LoPy.

Sur le côté gauche de la fenêtre pymakr, plusieurs icônes sont présentes :

- l'interrupteur permet d'activer ou de désactiver la connexion avec le LoPy ;
- le triangle permet d'exécuter le programme affiché dans la fenêtre d'Atom sur le LoPy ;
- la flèche vers le haut, permet de recopier le répertoire actif dans la mémoire du LoPy. Cela sera utile pour installer de nouveaux modules sur le LoPy ;
- inversement la flèche vers le bas, permet de recopier la mémoire du LoPy sur l'ordinateur ;
- le processeur permet d'avoir des informations sur le LoPy.

Sur la partie droite, l'onglet vertical *Setting* permet de modifier les paramètres de connexion avec le LoPy.

9.2.2 Installez votre environnement de travail

Pour programmer le LoPy, il faut récupérer les modules micropython. Le dépôt peut être téléchargé dans le répertoire de votre choix :

```
> git clone https://github.com/ltn22/PLIDObis.git
```

Dans le menu *Files>Open Folder* d'Atom, sélectionnez le répertoire pycom du dépôt téléchargé, et validez. Sur la partie gauche de l'écran, l'ensemble des fichiers composant ce répertoire apparaissent. Il y en a beaucoup, car ils vont nous servir par la suite.

En cliquant dans la fenêtre **pymakr** sur le bouton *Upload project to device*, les fichiers de ce répertoire vont être copiés dans la mémoire du LoPy. Par la suite, si un module est modifié, il devra être resynchronisé dans la mémoire du LoPy.

git et suivez les instructions).

3. Sous Linux, vous devez être membre du groupe dialout pour pouvoir gérer la communication sur le port USB. Si vous ne voyez pas l'invite, tapez `sudo adduser login dialout` en remplaçant login par le nom de votre compte Linux. Reconnectez-vous sous votre compte.

4. Les fenêtres sur fond gris montrent le code micropython et leur résultat.

9.3 Connexion au réseau Wi-Fi

Pour rattacher de LoPY à un réseau **Wi-Fi**, il doit dans un premier temps être configuré via la liaison USB de l'ordinateur pour lui donner les paramètres nécessaires à la connexion.

Le fichier `boot.py` a été copié lors du téléchargement des fichiers dans la mémoire du LoPy. Au démarrage du LoPy, ce programme va chercher à se connecter à un réseau Wi-Fi. Comme le nom du réseau et la clé secrète n'ont pas été fournie, il n'y arrive pas. Le LoPy se transforme en point d'accès. Au démarrage, le LoPy a dû afficher le message suivant, indiquant que le LoPy devient point d'accès Wi-Fi et va déployer son propre réseau sur lequel votre ordinateur peut se connecter. Le nom de ce réseau est de la forme `PLIDO_XXXX` où `XXXX` est une séquence hexadécimal propre à l'équipement. La clé est `www.pycom.io`. Le LoPy à l'adresse `192.168.4.1` sur ce réseau.

```
Failed to connect to any known network, going into AP mode
To connect look for 'PLIDO_5bac' access point, key = 'www.pycom.io'
```

Mais ce n'est pas très intéressant car votre ordinateur va perdre sa connexion à l'internet. Pas très pratique pour suivre le MOOC. Avant d'afficher ce message, le LoPy a montré la liste des réseaux Wi-Fi qu'il a détecté. Vous pouvez à l'inverse le connecter à un de ces réseaux en renseignant le fichier `wifi_conf.py` qui se trouve dans le répertoire `pycom`.

Listing 9.1 – `wifi_conf.py`

```
known_nets = {
    'MON_SSID': { 'pwd': 'MON_MOT_DE_PASSE' }
}
```

`MON_SSID` doit être remplacé par le nom du réseau Wi-Fi ou Service Set IDentifier (SSID) et `MON_MOT_DE_PASSE` par la clé qui y est associée. Notez que plusieurs réseaux Wi-Fi peuvent être ajoutés, puisque `MON_SSID` est vu comme une clé de l'objet JSON. L'édition de fichier s'est faite sur l'ordinateur, il doit être recopié dans la mémoire du LoPy en cliquant sur la flèche vers le haut.

Le Pycom redémarre et doit maintenant afficher un message du genre :

```
net to use ['MONWIFI']
Connected to MONWIFI with IP address: 192.168.1.76
Pycom MicroPython 1.20.2.r1 [v1.11-a5aa0b8] on 2020-09-09; LoPy4 with ESP32
Type "help()" for more information.
>>>
```

Il est possible de pinguer ou de se connecter avec **FTP** ou **telnet** en utilisant cette adresse IP.

```
# telnet 192.168.1.76
Trying 192.168.1.86...
Connected to 192.168.1.86.
Escape character is '^].
MicroPython v1.8.6-760-g90b72952 on 2017-09-01; LoPy with ESP32
Login as: micro
Password: python
Login succeeded!
Type "help()" for more information.
>>>
```

Youtube



Ça peut être utile pour suivre le comportement de votre objet sans lancer atom si l'objet n'est plus connecté via la liaison USB à l'ordinateur.

Atom peut également être configuré pour utiliser cette adresse IP. La configuration se fait dans le menu *setting*, et en entrant l'adresse ip du LoPy et en désactivant *auto connect*. Au prochain lancement d'Atom, il sera possible joindre le LoPy en Wi-Fi.

9.4 Mise en place d'un client

Un programme relativement simple permet de vérifier la communication entre le LoPy et le serveur. La commande **ifconfig**⁵ donne l'adresse IP du serveur. L'adresse doit être différente de celle que l'on avait obtenu sur le LoPy.

Si le serveur tourne dans un environnement local, l'adresse devrait commencer par 192.168 ou 10. Si le LoPy et l'ordinateur sont connectés au même réseau Wi-Fi, les premiers chiffres doivent être identiques.

Si le serveur tourne sur un serveur à l'extérieur (i.e. le *cloud*), l'adresse est quelconque.

La commande suivante est tapée depuis le terminal de l'ordinateur, on remarque les deux interfaces disponibles l'une pour le réseau Ethernet ou Wi-Fi et l'une pour le *loopback*, les noms des interfaces peuvent changer d'une configuration à une autre :

```
> ifconfig
eth1      Link encap:Ethernet HWaddr 10:65:30:b0:54:bf
          inet addr:192.168.1.237 Bcast:192.168.1.255 Mask:255.255.255.0
          inet6 addr: fe80::d8ac:86e7:8bdb:e333/64 Scope:Unknown
            UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
            RX packets:0 errors:0 dropped:0 overruns:0 frame:0
            TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0
            RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1 Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Unknown
            UP LOOPBACK RUNNING MTU:1500 Metric:1
            RX packets:0 errors:0 dropped:0 overruns:0 frame:0
            TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0
            RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)
```

Le programme `minimal_server.py` présenté au chapitre 7.1.1 page 84 n'a pas été modifié et attend des données sur le port 33033.

Listing 9.2 – sending_client.py

```
1 import socket
2
3 s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
```

5. Sous Linux, il faut ajouter le paquetage **net-tools** `sudo apt install net-tools`.



```
4 s.sendto("message", ("192.168.1.237", 33033))
```

Le programme `sending_client.py` doit être modifié (ligne 4) pour prendre en compte l'adresse IP du serveur obtenue avec la commande `ifconfig`. Si à chaque exécution sur le LoPy de ce programme, le serveur reçoit la valeur, la communication est établie entre les deux équipements.

```
> python3 minimal_server.py
b'message' => b'6d657373616765'
b'message' => b'6d657373616765'
```

9.5 BME 280

Au lieu de générer de fausses données, nous allons dans cette section utiliser un vrai capteur de température humidité pression : le BME 280 de chez Bosch.

9.5.1 Le bus I2C

Le bus I2C est normalisé par le fabricant de composants électroniques **NXP**⁶ ce qui permet une meilleure interopérabilité entre les composants.

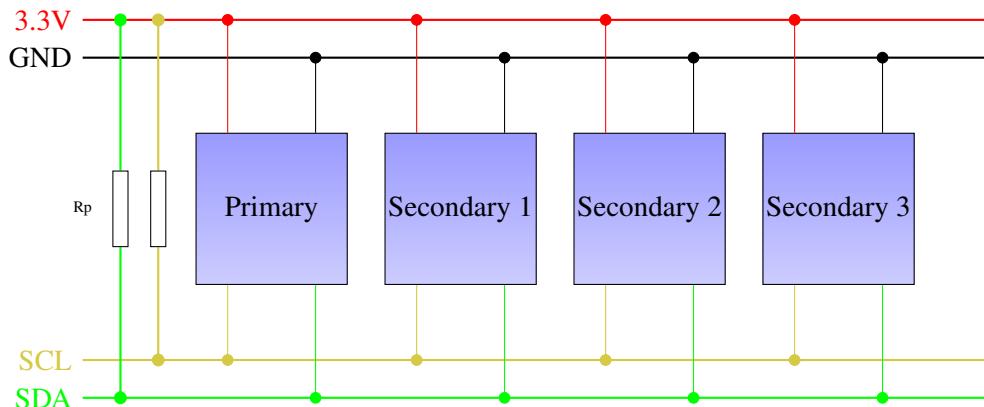


FIGURE 9.5 – bus I2C

Sur un des fils le signal d'horloge va être émis par le primaire. Sur l'autre fil, les données seront codées soit dans le sens primaire/secondaire, soit dans l'autre. Comme avec **Modbus**, les communications entre un secondaire et le primaire seront gérées par le primaire. Chaque secondaire est configuré avec une adresse unique sur le bus. Soit le maître envoie des données vers cette adresse⁷, soit le primaire interroge l'esclave pour obtenir ses données. Le fil sera donc exploité dans les deux directions.

La lecture de l'information binaire se fait lorsque le signal d'horloge est à l'état haut (cf. figure 9.6 page ci-contre). Quand le signal SDA est à l'état haut, un bit à 1 est transmis et dans l'état bas un bit

6. <https://www.nxp.com/docs/en/user-guide/UM10204.pdf>

7. Il s'agit d'un abus de langage, en effet les données émises par le primaire sont reçues par l'ensemble des secondaires, mais seul celui qui est destinataire (i.e. qui reconnaît son adresse) va les traiter, les autres équipements ignoreront l'information.

à 0 est transmis. Les changements d'état du signal SDA se font donc quand le signal d'horloge est à l'état bas.

Il existe malgré tout deux exceptions : si le signal SDA passe de l'état haut à l'état bas tandis que le signal d'horloge est à l'état haut cela indique un début de transmission de données. Si le signal SDA passe de l'état bas à l'état haut dans les mêmes conditions, cela indique la fin de transmission de données. Entre les deux les données binaires forment une trame (ou un PDU dans le vocabulaire ISO) qui est structuré comme indiqué figure 9.6.

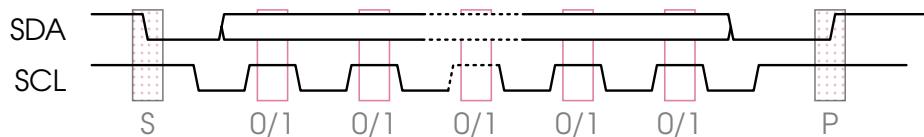


FIGURE 9.6 – Exemple de communication avec le bus I2C

La figure 9.7 donne les formats les plus utilisés.

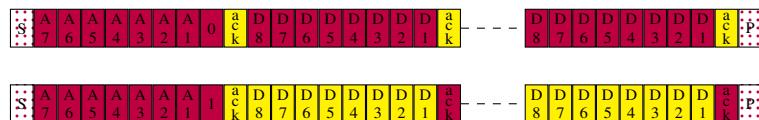


FIGURE 9.7 – Exemple de communication avec le bus I2C

Le premier train binaire illustre la transmission de données du primaire vers un secondaire. Le primaire commence par émettre un signal non binaire (S) indiquant un début de transmission. Les 7 bits suivants donnent l'adresse du secondaire et le bit suivant indique si le primaire veut envoyer des données (valeur à 0) ou recevoir des informations du secondaire (valeur à 1).

Si un secondaire reconnaît son adresse sur le bus, alors il écrit le bit suivant dans le train binaire. Le primaire est donc informé en lisant cette valeur que le secondaire est bien présent sur le bus et qu'il peut recevoir des données. Le primaire va donc les envoyer octet par octet. Chaque octet étant acquitté de la même manière par le secondaire. Le train binaire se termine par le signal non binaire (P).

Dans le cas où le primaire souhaite recevoir, une fois le bit de début et les bits de l'adresse émis, le huitième bit est positionné à 1. Le secondaire acquitte puis transmet ses octets que le primaire acquitte.

Question 9.5.1: scan

Le module I2C du LoPy dispose d'une fonction `scan` qui affiche les adresses des secondaires connectés. Comment cette détection est possible ?

Question 9.5.2: Diffusion

Est-ce que la norme une adresse qui permet de parler à tous les secondaires en même temps ?

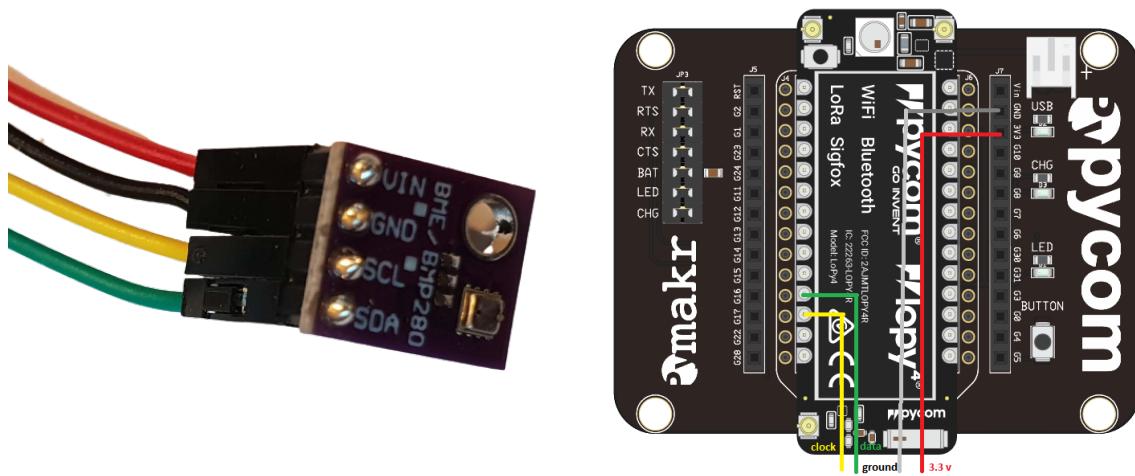


FIGURE 9.8 – capteur BME280 et connecteurs

9.5.2 Mesure de la température

La communication entre le LoPy et le composant se fait via le bus **I2C**. Il nous faut donc quatre fils pour le relier (cf. figure 9.8) :

- la masse (GND),
- une alimentation électrique de 3.3v (VIN),
- un fils pour l'horloge (SCL) et
- un autre finalement pour les données (SDA).

Pour ce faire, il faut connecter :

- la masse GND du LoPy sur la broche GND du composant avec un fil noir,
- l'alimentation en 3.3V du LoPy (3V3) sur VIN du composant avec un fil rouge (ce port s'appelle également **DCC** sur certaines cartes),
- Le signal d'horloge du port G18/P10 du LoPy (ou G17 sur les versions LoPy 1) sur le port SCL du composant avec un fil jaune (ce port s'appelle également **CLC** sur certaines cartes),
- Le fil de données du port G16/P9 du Pycom sur le port SDA du composant avec un fil vert.

Si le BME280 est connecté correctement sur les connecteurs du LoPy, vous devez obtenir le résultat suivant :

```
>>> Running BME280.py
>>>
>>>
[118]
temp 550576 27.98 - hum 26626 48.784 % - pres 389338 pres 1004.494 hPa [delta -389338 ]
temp 551952 28.42 - hum 26587 48.557 % - pres 389712 pres 1004.527 hPa [delta -374 ]
temp 551792 28.37 - hum 26577 48.491 % - pres 389664 pres 1004.621 hPa [delta 48 ]
temp 551712 28.34 - hum 26594 48.596 % - pres 389648 pres 1004.654 hPa [delta 16 ]
temp 551632 28.32 - hum 26587 48.546 % - pres 389616 pres 1004.713 hPa [delta 32 ]
```

Vous pouvez soit toucher le capteur, soit souffler dessus pour faire augmenter la température ou la pression.

Listing 9.3 – BME280.py

```
282 if __name__ == "__main__":
```

```

284     from machine import I2C
285     import time
286
287     i2c = I2C(0, I2C.MASTER, baudrate=400000)
288     print(i2c.scan())
289
290     bme = BME280(i2c=i2c)
291
292     ob = 0
293
294     while True:
295         ar = bme.read_raw_temp()
296         a = bme.read_temperature()
297
298         br = bme.read_raw_pressure()
299         b = bme.read_pressure()
300
301         cr = bme.read_raw_humidity()
302         c = bme.read_humidity()
303
304         print("temp", ar, a,
305               "hum", cr, c,
306               "%pres", br,
307               "pres", b,
308               "hPa[delta]", ob - br, "]")
309
310         time.sleep(5)

```

Le programme BME280.py peut fonctionner comme un module mais la dernière partie donne un exemple d'exploitation des résultats :

- Le programme principal commence par importer (ligne 283) le module gérant le bus I2C. Il est invoqué à la ligne 286. Le LoPy va gérer la communication avec le BME280, donc l'adresse est 0 et son statut estMASTER. La vitesse de communication est ensuite spécifiée.
- La ligne suivante scanne le bus pour trouver des composants. Si tout va bien, il devrait en trouver un à l'adresse 118 qui correspond à l'adresse par défaut du BME280⁸. Sinon, revoyez votre câblage.
- Le module BME280 est initialisé en lui passant en paramètre la référence du bus I2C précédemment défini.
- Le programme va ensuite afficher les valeurs captées par le composant. Il existe deux types de valeurs : brutes (*raw*) et calibrées. Les premières réagissent plus rapidement aux changements cependant sont beaucoup plus bruitées que les seconde qui subissent un traitement mathématique.
Ainsi, les première et deuxième colonnes donnent les températures brute et calibrée. On y accède par les méthodes `read_raw_temp` et `read_temperature`. Il en va de même pour les deux autres grandeurs, humidité et pression.
- Le programme affiche également l'écart de pression brute entre deux mesures. Cela permet de mettre plus facilement en évidence le fait que l'on souffle sur le capteur.

⁸. Si une autre valeur est indiquée, vous pouvez à l'instantiation du module BME280, ligne 289, ajouter le paramètre `addr=valeur`.

9.6 Thermomètre Wi-Fi

Vous avez maintenant tous les outils pour récupérer la température de votre logement, la transmettre à votre ordinateur via le Wi-Fi, et la transmettre à Beebotte pour l'afficher. Il y a très peu de changement par rapport à la version complètement sur ordinateur. Le programme de transformation de la structure CBOR de représentation des séries temporelles en JSON compréhensible par Beebotte reste le même. Il faudra juste modifier le nom du capteur d'humidité à température.

Le programme que vous avez construit lors du TP précédent utilisait le module `virtual_sensor` pour produire des séries temporelles aléatoires symbolisant le comportement d'un capteur. Maintenant que nous avons un **BME280**, nous allons pouvoir traiter de vraies valeurs.

Youtube



Le programme `wifi_temperature.py` montre cette adaptation.

Listing 9.4 – wifi_temperature.py

```

1 import BME280
2 import time
3 import socket
4 import kpn_senml.cbor_encoder as cbor
5 from machine import I2C
6
7
8 s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
9 NB_ELEMENT = 30
10 t_history = []
11
12 i2c = I2C(0, I2C.MASTER, baudrate=400000)
13 print(i2c.scan())
14
15 bme = BME280.BME280(i2c=i2c)
16
17 while True:
18
19     t = int(bme.read_temperature()*100)
20
21     # No more room to store value, send it.
22     if len(t_history) == 0:
23         t_history = [t]
24     elif len(t_history) >= NB_ELEMENT:
25         print("send")
26         s.sendto(cbor.dumps(t_history), ("192.168.1.47", 33033))
27         t_history = [t]
28     else:
29         t_history.append(t)
30
31     prev = t
32
33     print(len(t_history), len(cbor.dumps(t_history)), t_history)
34
35     time.sleep(10)

```

Au niveau des importations de modules, BME280 et remplacent `virtual_sensor`, et le module CBOR est celui de `kpn_senml`.

Mais son comportement reste le même, en particulier la fonction `dumps` qui convertit une structure Python en CBOR. Il vous reste à adapter la ligne 26 pour mettre l'adresse IP de votre ordinateur.

Côté ordinateur, vous devez relancer le programme `display_server.py`, mais en modifiant le nom du capteur de "humidity" à "temperature".

Sur votre compte Beebotte, au bout de 300 secondes, vous devez voir des données associées au capteur "temperature".

Question 9.6.1: changement de pas

Que se passe-t-il si dans le programme `wifi_temperature.py` vous modifiez le pas de mesure ligne 36, pour le mettre par exemple à 60 secondes.

10. Sigfox

Sigfox est l'un des tous premiers réseaux entièrement dédié à l'Internet des Objets. Comme nous l'avons vu auparavant, il est de la famille des **LPWAN** qui privilégie la portée et la consommation d'énergie au débit. Les communications sont également fortement asymétrique, ce qui fait que les échanges ne sont pas comme sur un réseau Wi-Fi.

Les LoPy peuvent utiliser le réseau et bénéficie d'un an de connectivité gratuite sur le réseau Sigfox. Après les coûts d'abonnement sont relativement limités.

Youtube



10.1 Récupération des identifiants

Dans un premier temps, vous devez enregistrer votre capteur sur le site de Sigfox. Il vous faut deux éléments : son identifiant et son mot de passe appelé Porting Authorization Code (PAC). Ce dernier doit rester secret car il permet à toute personne qui le possède d'enregistrer un objet ou d'en changer le propriétaire.

Le programme `sigfox_id.py` permet d'afficher ces deux valeurs et d'envoyer un message sur le réseau Sigfox. Avant de l'exécuter, vérifiez que vous avez branché une antenne sur le connecteur de droite, celui opposé au bouton Reset sur le Pycom, cote LED.

Listing 10.1 – `sigfox_id.py`

```
1 from network import Sigfox
2 import binascii
3 import socket
4
5 # initialise Sigfox for RCZ1 (You may need a different RCZ Region)
6 # RCZ1: Europe, Oman, South Africa
7 # RCZ2: USA, Mexico, Brazil
8 # RCZ3: Japan
9 # RCZ4: Australia, New Zealand, Singapore, Taiwan, Hong Kong, Columbia, Argentina
```

```

11  sigfox = Sigfox(mode=Sigfox.SIGFOX, rcz=Sigfox.RCZ1)
12
13 # print Sigfox Device ID
14 print("Sigfox_ID:", binascii.hexlify(sigfox.id()))
15
16 # print Sigfox PAC number
17 print("PAC_Number:", binascii.hexlify(sigfox.pac()))
18
19 s = socket.socket(socket.AF_SIGFOX, socket.SOCK_RAW)
20 s.send("Hi!_Sigfox")

```

Ce programme importe l'objet Sigfox du module `network` (ligne 1) et crée une instance `sigfox` (ligne 10). Il est important de bien spécifier la bonne région d'utilisation car les bandes de fréquences peuvent différer d'un continent à un autre. En plus d'émettre dans l'illégalité, le réseau Sigfox ne recevra pas les messages.

Les lignes 13 et 16 affichent les identifiants Sigfox de votre LoPy. Notez-les, ils nous serviront pour enregistrer l'objet sur le réseau de Sigfox.

La ligne 18 crée une `socket`, à l'instar de ce qui avait été fait avec UDP au chapitre précédent. On peut donc utiliser les mêmes primitives en Sigfox qu'en UDP. La ligne 19 permet d'envoyer un message que vous pouvez personnaliser dans la limite de 12 caractères ; taille maximale des trames Sigfox.

10.2 Enregistrement de l'objet

Maintenant que vous avez les précieux identifiants Sigfox, connectez-vous avec un navigateur sur le site <https://backend.sigfox.com/activate>. Le processus est très simple. Il suffit de remplir les champs des différents formulaires :

- indiquez votre pays ;
- remplissez le formulaire avec l'identifiant de l'objet et le **PAC** que vous avez obtenus avec le programme `sigfox_id.py` ;
- indiquez à des fins de statistique ce qui vous amène ici ;
- créez votre compte Sigfox.

Ce qui va conduire à enregistrer l'objet et à vous créer un compte sur le backend de Sigfox.

10.3 Visualisation des données émises par le Pycom

Rendez-vous sur le **backend** de Sigfox (<https://backend.sigfox.com/>) et identifiez-vous avec le compte que vous venez de créer.

Les onglets en haut de la page (cf. figure 10.1 page suivante) vont vous permettre de naviguer dans différents types d'information. Dans cette partie, nous n'utiliserons que l'onglet *DEVICE*. Il donne accès aux objets enregistrés vous appartenant (cf. figure 10.2 page suivante). On y retrouve :

- un nom créé par Sigfox en fonction du type d'objet ;
- le propriétaire ;
- l'ID que vous avez donnés lors de l'enregistrement ;
- la date du dernier message reçu par Sigfox pour cet objet.

Il faut cliquer :

- sur le nom de l'objet pour le configurer ;

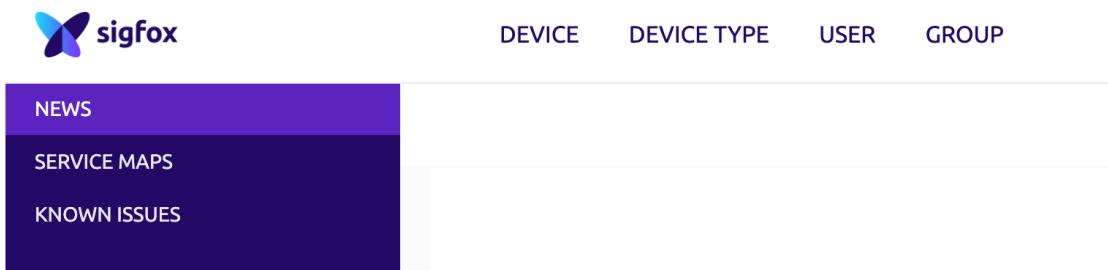


FIGURE 10.1 – Page d'accueil

page 1							
Communication status	Device type	Group	Id	Last seen	Name	Token state	
<input checked="" type="radio"/>	PYCOM_DevKit_1	IMT Atlantique	4D55AC	2020-08-12 16:22:07	PYCOM_DevKit_1-device	<input checked="" type="checkbox"/>	
page 1							

FIGURE 10.2 – Objets enregistrés

- sur le nom du groupe pour accéder a des paramètres d'administration des objets ;
- sur l'ID de l'objet pour obtenir des informations concernant les messages reçus puis *MESSAGES* dans le menu de gauche, pour avoir la liste des messages reçus par Sigfox, comme montré à la figure 10.3 page ci-contre¹.

10.4 Que s'est-il passé coté radio

La figure 10.4 page suivante montre sur un **analyseur de spectre**, l'émission de 4 trames de données, dont une en cours, par un objet. Les petits traits verticaux correspondent à une émission. Ces traits sont très fin ; la bande passante utilisée est très petite, d'où le terme anglais de Ultra Narrow-Band (UNB). Cela limite le risque de **collision** qui rendrait la donnée incompréhensible lié à l'émission simultanée d'un autre équipement sur la même fréquence.

En fait, le même message est émis 3 fois sur des fréquences différentes et aléatoire, augmentant ses chances d'être reçu.

10.5 Récupération des données

L'idéal serait d'avoir directement accès a ces données pour pouvoir les manipuler dans un programme. Pour ce faire, nous pouvons utiliser l'API REST développée par Sigfox. Dans l'onglet *DEVICE* (figure 10.2), il faut cliquer cette fois ci sur :

- le nom de votre groupe ;
- dans le menu de gauche, sur API ACCESS ;
- en haut à droite, sur le tout petit bouton *New*.



1. La séquence 48 69 21 20 53 69 67 66 6f 78 correspondant à la chaîne de caractères Hi ! Sigfox.

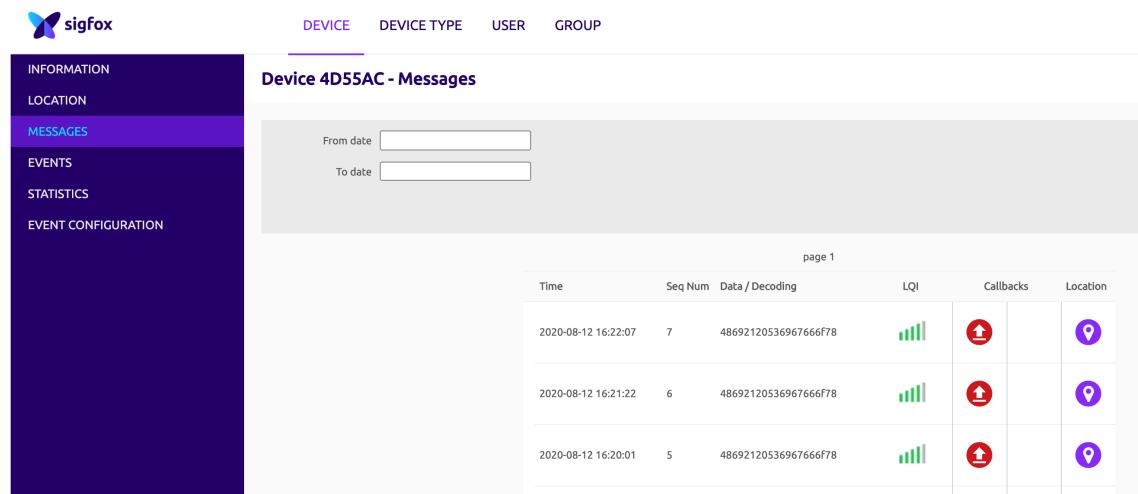


FIGURE 10.3 – Liste des messages reçus

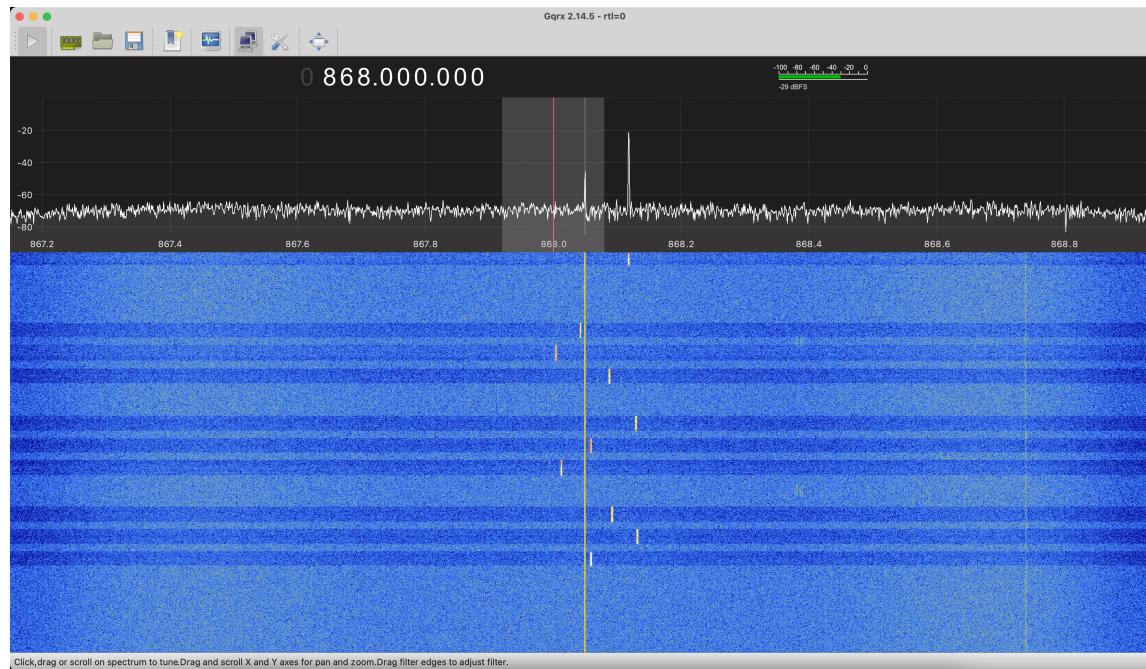


FIGURE 10.4 – Émissions radio liées à l'envoi de trame Sigfox.

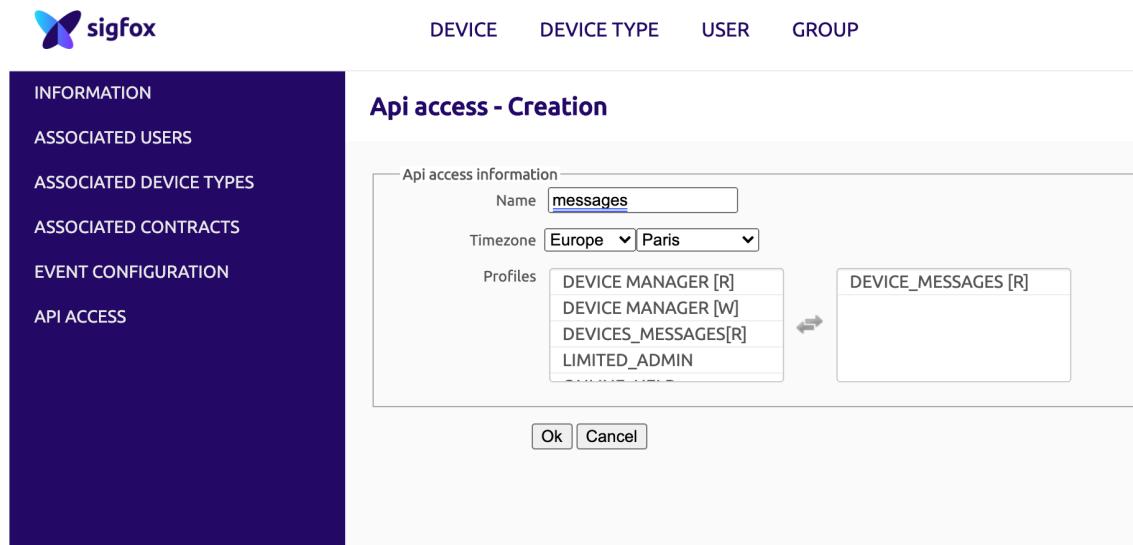


FIGURE 10.5 – Configuration de l’API REST.

Une page similaire à la figure 10.5 s’affiche. Donnez un nom à cet accès et choisissez, dans le menu *Profiles* le choix *DEVICE MESSAGE [R]* pour avoir le droit de lire les messages. Puis évidemment, sur Ok.

Vous voyez apparaître une nouvelle page avec deux champs en hexadécimal : *login* et *password*, que vous devez, comme pour l’API de Beebotte, noter quelque part ou apprendre par cœur pour la suite.

Le mieux étant de remplir un fichier de configuration avec ces valeurs comme le montre le programme config_sigfox.py.

Listing 10.2 – config_sigfox.py

```
1 API_USER = "603f575525643207b6322e9b"
2 API_PASSWORD = "93fa105063c2d0aeee2bfdbadccfd2460"
3 DEVICEID = "1B28CF4"
```

Tous les éléments sont maintenant réunis pour émettre un relevé de température en utilisant le réseau Sigfox.

10.5.1 Sur le serveur

Nous avons les clés d'accès à l'API, il suffit maintenant d'écrire un petit script Python.

Pour des raisons de sécurité, nous vous invitons à prendre l'habitude de mettre les informations sensibles dans un fichier séparé.

Le programme device_messages.py permet de lister les messages reçus par Sigfox pour un objet particulier sur votre ordinateur.

Listing 10.3 – device_messages.py

```
1 import requests
2 import os
```

```

3 from requests.auth import HTTPBasicAuth
4 import pprint
5 import json
6 from config_sigfox import API_USER, API_PASSWORD, DEVICEID
7 import binascii

9 url = 'https://backend.sigfox.com/api/v2/devices/' + DEVICEID + '/messages'

11 print(url)

13 r = requests.get(url, auth=HTTPBasicAuth(API_USER, API_PASSWORD))
14 print(r.status_code)

15 if r.status_code != 200:
16     exit

19 resp = json.loads(r.text)
20 pprint.pprint(resp)
21 for v in resp["data"]:
22     print ("{:10d} {:2d} {:25} {:20} received {}".format(
23         v["time"], v["seqNumber"],
24         v["data"], "[" + str(binascii.unhexlify(v["data"])) + "]",
25         len(v["rinfos"])))
26

```

Le programme :

- Ligne 1, importe le module `requests` pour pouvoir envoyer des requêtes http à un serveur.
- Ligne 3, le module `HTTPBasicAuth` est utilisé pour s'identifier de façon simple en utilisant un login et un mot de passe.
- Ligne 6 ce login et ce mot de passe sont extrait du fichier rempli au chapitre précédent lors de la création de l'API REST.
- Ligne 9, l'URL comportant l'ID de l'objet est construite et
- ligne 13 la requête HTTP est envoyée avec la méthode d'authentification basée sur le mot de passe. la variable `r` est une structure contenant plusieurs informations.
- Lignes 14 à 17, Si le code retourne est 200, tout s'est bien passé et `r.text` contient la réponse.
- Ligne 19, cette réponse est une chaîne de caractères qui est désérialisée de la représentation JSON pour en faire une structure Python grâce à la fonction `loads`. Line 19, this response is a string that is serialized from the JSON representation to a Python structure using the function `loads`.
- Ligne 20, la réponse est affichée et ensuite certains éléments sont donnés. On y retrouve tous les messages qui ont été émis par le LoPy.

```

>python3 device_messages.py
https://backend.sigfox.com/api/v2/devices/1B28CF4/messages
200
{'data': [{}{'country': 'FRA',
            'data': '48692120536967666f78',
            'device': {'id': '1B28CF4'},
            'lqi': 3,
            'nbFrames': 3,
            'operator': 'SIGFOX_France',
            'rinfos': []},

```

```

        'rolloverCounter': 0,
        'satInfos': [],
        'seqNumber': 13,
        'time': 1640279367000},
    {'country': 'FRA',
     'data': '48692120536967666f78',
     ...
     'rolloverCounter': 0,
     'satInfos': [],
     'seqNumber': 11,
     'time': 1640279155000}],
'paging': {}}
1640279367000: 13 48692120536967666f78 [b'Hi! Sigfox'] received 0
1640279338000: 12 48692120536967666f78 [b'Hi! Sigfox'] received 0
1640279155000: 11 48692120536967666f78 [b'Hi! Sigfox'] received 0

```

La fin de la trace affiche un résumé plus lisible de l'information reçue :

- `time` donne l'heure de réception codée suivant le format **Epoch**, évoqué lors de la communication avec Beebotte au chapitre précédent;
- `seqNumber` contient le numéro de la trame et est remis à 0 quand l'objet est reflashe. Il permet de détecter des pertes de données si les numéros ne sont pas contigus;
- "data" contient les données codées dans une chaîne hexadécimale. Le programme utilise la fonction `unhexify` pour la reconvertis en séquence d'octets qui peuvent être affichés s'il s'agit de caractères ASCII;
- `rinfos` donne les informations sur les différentes passerelles radio de l'opérateur qui ont reçu le message.

10.5.2 Sur le LoPy

Le programme `sigfox_temperature.py` est une adaptation de `wifi_temperature.py`, listing 9.6 page 114 dédié au Wi-Fi pour des transmission sur le réseau Sigfox.

Listing 10.4 – `sigfox_temperature.py`

```

1 import BME280
2 import time
3 import socket
4 import kpn_senml.cbor_encoder as cbor
5 from machine import I2C
6 from network import Sigfox
7 import binascii
8 import socket
9
# initialise Sigfox for RCZ1 (You may need a different RCZ Region)
10 sigfox = Sigfox(mode=Sigfox.SIGFOX, rcz=Sigfox.RCZ1)
11 s = socket.socket(socket.AF_SIGFOX, socket.SOCK_RAW)
12
13 FRAME_MAX = 12
14 t_history = []
15
16 i2c = I2C(0, I2C.MASTER, baudrate=400000)
17 print(i2c.scan())
18 bme = BME280.BME280(i2c=i2c)

```

```

21 while True:
22
23     t = int(bme.read_temperature()*100)
24
25     # No more room to store value, send it.
26     if len(t_history) == 0:
27         t_history = [t]
28     else:
29         t_history.append(t-prev)
30
31     print (t_history, len(cbor.dumps(t_history)))
32
33     if len(cbor.dumps(t_history)) > FRAME_MAX:
34         # oops too big for Sigfox
35         t_history = t_history[:-1] # remove last item
36         s.send (cbor.dumps(t_history))
37         t_history = [t]
38
39     prev = t
40
41     time.sleep(10)

```

- ligne 5, la classe Sigfox du module network est installée ;
- ligne 11, un objet texttt est instancié avec, ici, les paramètres pour l’Europe ;
- ligne 12, au lieu de faire appel à AF_INET pour utiliser la pile protocolaire TCP/IP, la valeur AF_SIGFOX est utilisée.
- ligne 14, la taille de la trame est fixée à 12 octets pour être compatible avec le réseau.

Le programme s’exécute sur le LoPy.

```

>>> Running sigfox_temperature.py
>>>
>>>
[118]
[2192] 4
[2192, -89] 6
[2192, -89, -16] 7
[2192, -89, -16, -12] 8
[2192, -89, -16, -12, -15] 9
[2192, -89, -16, -12, -15, -23] 10
[2192, -89, -16, -12, -15, -23, -11] 11
[2192, -89, -16, -12, -15, -23, -11, -14] 12
[2192, -89, -16, -12, -15, -23, -11, -14, -13] 13
[1999, -12] 5
[1999, -12, -5] 6
[1999, -12, -5, -8] 7
[1999, -12, -5, -8, -9] 8
[1999, -12, -5, -8, -9, -6] 9
[1999, -12, -5, -8, -9, -6, 2] 10
[1999, -12, -5, -8, -9, -6, 2, 0] 11
[1999, -12, -5, -8, -9, -6, 2, 0, -5] 12
[1999, -12, -5, -8, -9, -6, 2, 0, -5, -2] 13
[1954, -4] 5
[1954, -4, -1] 6

```

Le programme device_messages.py récupère également ces valeurs depuis l’ordinateur.

1640281923000: 15 891907cf2b24272825020024	[b"\x89\x19\x07\xcf+\$'(\x02\x00\$"] received 0
1640281823000: 14 8819089038582f2b2e362a2d	[b'\x88\x19\x08\x908X/+.*-'] received 0
1640279367000: 13 48692120536967666f78	[b'Hi! Sigfox'] received 0
1640279338000: 12 48692120536967666f78	[b'Hi! Sigfox'] received 0
1640279155000: 11 48692120536967666f78	[b'Hi! Sigfox'] received 0

Attention suivant les variations de la température, le message CBOR grandit plus au moins vite. Dans le cas précédent, il y avait une émission toutes les 90 secondes, or l’abonnement au réseau Sigfox limite le nombre d’émission à 140 messages par jours. Au bout de 3 heures le quota de

message sera épuisé. Cette petite période d'émission permet de tester plus rapidement les solutions, mais il convient d'augmenter la période pour une utilisation régulière.

10.5.3 requête GET depuis le serveur

Côté ordinateur, la stratégie la plus simple à mettre en œuvre consiste à étendre le programme `device_message.py` vu précédemment et d'interroger périodiquement le *backend* de Sigfox pour voir si de nouvelles données sont arrivées. Cela donne le programme `display_sigfox.py` suivant :

Listing 10.5 – `display_sigfox.py`

```

1 import requests
2 import os
3 from requests.auth import HTTPBasicAuth
4 import pprint
5 import json
6 from config_sigfox import API_USER, API_PASSWORD, DEVICEID
7 import binascii
8 import time
9 import datetime
10 import cbor2 as cbor
11 import beebotte
12 import config_bbt
13
14 bbt = beebotte.BBT(config_bbt.API_KEY, config_bbt.SECRET_KEY)

```

Les importations incluent les modules pour envoyer les données à Beebotte et pour interroger Sigfox. Ligne 14, la communication avec le serveur de Beebotte est établie en utilisant les secrets du module `config_btt`.

```

url = 'https://backend.sigfox.com/api/v2/devices/' + DEVICEID + '/messages'
last_epoch = 0

# get the last message to find the starting epoch
parameters = {'limit': 1}
r = requests.get(url, auth=HTTPBasicAuth(API_USER, API_PASSWORD),
                  params=parameters)

print (r.status_code)
if r.status_code != 200:
    exit

j = json.loads(r.text)
last_epoch = j["data"][0]["time"]

```

la fonction `to_bbt` n'a pas été modifiée, on passe donc à la récupération des données sur les serveurs de Sigfox. Le programme va interroger régulièrement le serveur de Sigfox pour récupérer les nouveaux messages. Comme Sigfox stocke l'ensemble des messages reçus, cela peut conduire à un trafic conséquent. Pour limiter le trafic, le programme n'affichera que les nouvelles données qui arrivent pendant son exécution.

Le programme :

- ligne 41, construit l'URL pour la requête en incluant l'identificateur du LoPy ;
- ligne 42, initialise la variable `last_epoch`. Elle contient l'epoch du dernier message reçu par Sigfox pour cet objet ;

- ligne 45, une structure JSON est créée, elle contient la clé `limit` et la valeur 1, pour indiquer à Sigfox que l'on souhaite recevoir en réponse que le dernier message reçu ;
- lignes 46 à 51, la requête est envoyée avec le paramètre précédemment défini. Si le status n'est pas 200, le programme s'arrête.
- lignes 53 et 54, la variable `last_epoch` reçoit l'instant d'arrivée du dernière message.

```
56 # delay next request to avoid 429 status error
57 time.sleep(10)
```

Un délai de 10 secondes est introduit entre deux requêtes, car Sigfox limite le nombres de requête pour éviter la saturation de ses serveurs.

```
# look periodically if new data arrived.
60 while True:
61     if last_epoch > 0:
62         parameters = {"since": last_epoch+1}
63     else:
64         parameters = None
65
66     print (parameters)
67
68     r = requests.get(url, auth=HTTPBasicAuth(API_USER, API_PASSWORD),
69                       params=parameters)
70
71     print (r.status_code)
72     if r.status_code != 200:
73         break
```

Le programme entre dans une boucle sans fin où il va interroger régulièrement le serveur Sigfox. Les interrogations sont identiques à la précédente, mais le paramètre contient un objet JSON avec la clé `since` et l'`epoch`. Si la réponse à la requête n'est pas 200: OK, le programme s'arrête.

```
76     resp = json.loads(r.text)
77
78     for v in resp["data"]:
79         if last_epoch < v["time"]:
80             last_epoch = v["time"]
81
82         print ("{:10d}:{:2d}{:25}{:20}received{}".format(
83             v["time"], v["seqNumber"],
84             v["data"], "["+str(binascii.unhexlify(v["data"]))+"]",
85             len(v["rinfos"]))
86
87         measure = cbor.loads(binascii.unhexlify(v["data"]))
88         sending_time = v["time"]
89         print (sending_time, measure)
90         to_bbt("capteurs", "temperature", measure, factor=0.01,
91                period=10, epoch=sending_time)
```

La structure JSON reçue est déserialisée ligne 75 pour en faire un tableau de messages enrichis de paramètres ajoutés par Sigfox. Pour chacun de ces éléments qui correspondent aux nouveaux messages reçus, Le programme va faire avancer la variable `last_epoch` sur la plus grande valeur (lignes 78 et 79), puis ligne 87 va prendre les données indiquées par la clé `data` dans le dictionnaire. Ces données, correspondent au tableau CBOR envoyé par l'objet, codés en une chaîne de caractère

hexadécimaux. La fonction `unhexlify` la convertie en sequence binaire dans une chaîne d'octet puis `loads` transforme le tableau CBOR en un tableau Python.

Ligne 91 et 92, la fonction `to_btt` (ligne 16 à 39 non représenté dans ce listing) est appelée. Elle est identique à celle qui avait été présentée au chapitre précédent. Mais comme la procédure est asynchrone, le programme traite les données quand il en fait la demande, pas quand le capteur émet des données, le timestamp doit être celui indiqué par Sigfox, il est passé dans le paramètre `epoch`.

```
time.sleep(60)
```

Le programme attend 60 seconde avant de faire une nouvelle requête.

10.5.4 requête POST vers le serveur

Nous avons dû modifier la logique du programme `display_sigfox.py`. Au lieu d'attendre des données comme le faisait `display_server.py` en Wi-Fi, il va chercher activement les données sur le *backend*. Cela est imposé par les limitations de l'adressage IPv4 privé que vous avez certainement sur votre réseau local. En effet, il n'est pas possible d'être joint par l'extérieur, les connexions ne se font qu'à l'initiative de l'équipement qui a une adresse privée. Pour revenir au cas précédent où le *backend* pourrait pousser des données, on peut soit faire tourner le programme sur une machine virtuelle dans le Cloud (ex : Virtual Private Server (VPS) d'OVH) ou configurer le Network Address Translation (NAT) de votre boîtier Internet pour autoriser des connexions entrantes sur un port particulier. C'est cette dernière option que nous allons détailler. Bien entendu, les caractéristiques des NAT changent d'un opérateur à un autre ; nous ne pourrons pas détailler sa configuration.



Configuration du callback

Dans un premier temps, rendez-vous sur <https://backend.sigfox.com/device/list>, cliquez sur le nom de l'objet puis, dans le menu de gauche, sur *Callback*, puis *New*. Une page apparaît avec la possibilité de connecter l'objet à différentes plateformes. Nous choisissons *Custom callback* car nous voulons garder notre indépendance. La figure 10.6 page suivante montre le formulaire.

Ensuite, il faut déterminer l'adresse IP publique derrière votre NAT². L'URI de votre serveur sera quelque chose comme indiquée ci-dessous pour l'*Url Pattern*, où *aaa.bbb.ccc.ddd* représente l'adresse IP publique³.

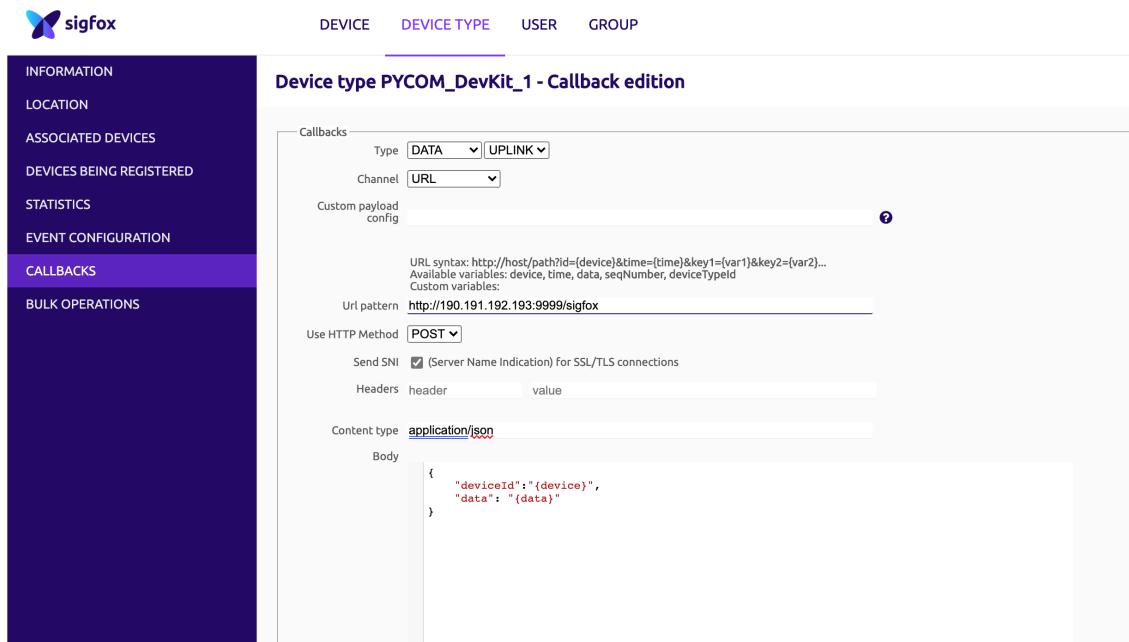
```
http://aaa.bbb.ccc.ddd:9999/sigfox
```

Changez la méthode de **GET** par un **POST**, c'est quand même plus propre. Le GET étant un moyen de récupérer de l'information pas d'en transmettre si on veut rester compatible avec REST.

Ce n'est pas fini. Il faut maintenant formater le contenu. Dans Content type, remplacez la valeur par `application/json` car c'est ce que l'on sait faire de mieux. Dans la fenêtre en dessous, nous allons définir notre format JSON avec deux clés : `deviceId` et `data`, suivies de deux variables entre accolades que Sigfox remplacera par les vraies valeurs.

2. Un site comme <https://wtfismyip.com/> peut aider.

3. Attention, certains opérateurs changent régulièrement l'adresse IP allouée au NAT. Il est donc préférable d'utiliser un DNS dynamique si vous voulez l'utiliser à long terme.

FIGURE 10.6 – Configuration d'un *callback*.

Finalement, cliquez sur Ok. Maintenant, quand Sigfox recevra des données de l'objet, il enverra une requête POST à l'URI indiquée.

Configuration du NAT

Il vous reste à configurer le NAT de votre routeur d'accès pour que les paquets à destination du port 9999 (valeur choisie dans l'URI) TCP soient envoyés à l'adresse privée de votre ordinateur. La démarche est généralement la même : configurer DHCP pour que votre ordinateur ait toujours la même adresse dans la maison puis configurer le NAT pour que les paquets adressés à un port soit envoyé à cet ordinateur.⁴

Wireshark peut vérifier que les requêtes traversent bien le NAT et arrive à votre ordinateur en regardant le trafic TCP sur le port 9999.

Traitement des requêtes POST

Les données contenues dans la requête POST venant du *backend* de Sigfox seront transférer sur le lien *loopback* de l'ordinateur vers le port UDP 33033. Le programme `display_server.py` attendant les données sur ce port pourra être utilisé pour transformer la série temporelle codée en CBOR en structure JSON attendue par Beebotte. De cette manière, nous banalisons le programme qui pourra traiter ces trois sources d'information :

- objet émulé par un programme Python,
- LoPy avec Wi-Fi,
- LoPy avec Sigfox,
- et bientôt LoPy avec LoRaWAN.

4. Si vous ne pouvez pas le faire, vous pouvez casser votre tirelire pour vous acheter un VPS avec une adresse IP publique pour quelques par mois. Dans ce cas, il faudra installer Python3 et les modules nécessaires (Beebotte, cbor2...)

Le programme `generic_relay.py` le fait pour Sigfox et différents serveurs LoRaWAN. Nous ne dévoilerons ici que les lignes liées au traitement de Sigfox.

Listing 10.6 – `generic_relay.py`

```

54 def get_from_sigfox():

56     fromGW = request.get_json(force=True)
57     print ("SIGFOX POST RECEIVED")
58     pprint.pprint(fromGW)

59     downlink = None
60     if "data" in fromGW:
61         payload = binascii.unhexlify(fromGW["data"])
62         downlink = forward_data(payload)

63     resp = Response(status=200)
64     print (resp)
65     return resp
66

```

On se rappelle du serveur **Flask** que l'on avait vu au chapitre 3.4 page 48, dans ce programme on retrouve les mêmes principes :

- ligne 54, le decorateur permet de lier la fonction `get_from_sigfox` au chemin d'URI `/sigfox` pour la methode POST.
- ligne 57 déserialise le contenu du POST qui est contenu dans la variable `request`. Le paramètre `force` est là au cas où vous auriez oublié de mettre sur le *backend* le format du contenu à `application/json`. Tout ce qu'il reçoit est considéré comme du JSON.
- lignes 62 à 64, si le POST contient des données (clé `data` dans l'objet JSON, alors elles sont envoyées sur le lien *loopback* via la fonction `forward_data`.
- lignes 66 à 68, le POST est acquitté avec le status 200 pour indiquer que le traitement de la requête s'est bien déroulé.

A noter que la fonction `forward_data` peut retourner une réponse qui sera envoyée au client HTTP. Pour Sigfox, cette possibilité n'est pas prise en compte car Sigfox n'autorise que 4 messages descendant par jour. Nous détaillerons son fonctionnement lorsque nous étudierons les réseaux LoRaWAN⁵.

```

192 parser = argparse.ArgumentParser()
193 parser.add_argument("-v", "--verbose",
194                     action="store_true",
195                     help="show uplink and downlink messages")
196 parser.add_argument('--http_port', default=9999,
197                     help="set http port for POST requests")
198 parser.add_argument('--forward_port', default=33033,
199                     help="port to forward packets")
200 parser.add_argument('--forward_address', default='127.0.0.1',
201                     help="IP address to forward packets")

202 args = parser.parse_args()
203 verbose = args.verbose
204 defPort = args.http_port
205 forward_port = args.forward_port

```

5. Pour plus de détails sur l'envoi de messages descendant voir page 143.

```
208     forward_address = args.forward_address
      app.run(host="0.0.0.0", port=defPort)
```

La partie principale du programme analyse les arguments utilisés lors de son appel. Si l'option `-v` est utilisé, le programme affichera les messages relayés. D'autres options sont définies pour modifier les numéro de port. Ces dernières sont utiles si ces programmes sont lancés sur un même machine dans le cloud.

Exemple

L'exemple suivant trace le parcours d'une série temporelle depuis un LoPy jusqu'à son envoie au serveur Beebotte pour visualisation.

```
>>> Running sigfox_temperature.py

>>>
>>>
[118]
[1895] 4
[1895, -4] 5
[1895, -4, -2] 6
[1895, -4, -2, 1] 7
[1895, -4, -2, 1, -3] 8
[1895, -4, -2, 1, -3, -1] 9
[1895, -4, -2, 1, -3, -1, 0] 10
[1895, -4, -2, 1, -3, -1, 0, 0] 11
[1895, -4, -2, 1, -3, -1, 0, 0, 6] 12
[1895, -4, -2, 1, -3, -1, 0, 0, 6, 2] 13
```

Le LoPy collecte les mesures et construit sa série temporelle qui est envoyée quand la capacité de la trame est atteinte. Dans le cas de Sigfox, il s'agit de 12 octets, la dernière ligne montre que la capacité est dépassée, donc le dernier élément est retiré.

```
>python3.5 generic_relay.py -v
SIGFOX POST RECEIVED
{'data': '891907672321012220000006', 'deviceId': '4D3D0E'}
--UP-> b'891907672321012220000006',
no DW
<Response 0 bytes [200 OK]>
185.110.98.2 - - [24/Dec/2021 10:14:26] "POST /sigfox HTTP/1.1" 200 -
```

Le programme `generic_relay.py` reçoit la requête POST du réseau Sigfox sur le chemin d'URI `/Sigfox`. Elle contient l'élément CBOR envoyé par le LoPy. Ces données sont envoyées sur le port 33033. Aucune réponse n'est reçue, la requête POST est simplement acquittée.

```
>python3 display_server.py
[{'data': 18.95, 'resource': 'temperature', 'ts': 1640337186000.0},
 {'data': 18.91, 'resource': 'temperature', 'ts': 1640337196000.0},
 {'data': 18.89, 'resource': 'temperature', 'ts': 1640337206000.0},
 {'data': 18.90, 'resource': 'temperature', 'ts': 1640337216000.0},
 {'data': 18.87, 'resource': 'temperature', 'ts': 1640337226000.0},
 {'data': 18.86, 'resource': 'temperature', 'ts': 1640337236000.0},
 {'data': 18.86, 'resource': 'temperature', 'ts': 1640337246000.0},
```

```
{'data': 18.86, 'resource': 'temperature', 'ts': 1640337256000.0},  
{'data': 18.92, 'resource': 'temperature', 'ts': 1640337266000.0}]
```

Le programme `display_server.py` traite la donnée CBOR reçue sur le port 33033 et la convertie dans le format JSON attendu par Beebotte.

10.6 Conclusion

On a construit un prototype de capteur qui fonctionne sur Sigfox. À vous de l'améliorer. En particulier, il faut augmenter l'intervalle entre deux mesures qui a été fixe à 10 secondes pour ne pas avoir à faire des tests trop longs. Comme la taille d'un message est de 12 octets, CBOR va prendre 1 octet pour coder le tableau et la valeur de référence est codée sur 3 octets. Si les deltas sont petits, ils tiendront sur un octet. Il reste de la place pour 8 deltas. Donc, la période d'émission est de 90 secondes si les intervalles entre deux mesures restent à 10 secondes.

Comme Sigfox n'autorise que 140 messages par jours, la durée de mesure ne sera que de 210 minutes par jours, soit 3 heures et demie. Il est donc préférable de prendre des intervalles plus grands. Vous pouvez calibrer votre LoPy et votre programme de réception pour pouvoir suivre la température sur une journée. N'oubliez pas de changer également cette période dans le programme `display_server.py`.

Le programme `display_server.py` permet de récolter des informations de plusieurs sources :

- un programme en local qui émule les mesures,
- du réseau Wi-Fi,
- par l'intermédiaire de `generic_relay.py` provenant d'un réseau LPWAN.

Nous avons notre convention de représentation de l'information basée sur CBOR pour définir une série temporelle. En revanche, nous avions été obligés de modifier le programme `display_server.py` quand nous étions passé d'une série temporelle représentant l'évolution de l'humidité à une autre traitant de la température et si la période change.

Nous avons vu la différence entre les requêtes GET et POST qui induisent des comportements différents. Si GET est plus universel et peut fonctionner derrière un NAT, il demande des interrogations régulières pour savoir si la ressource a changé ou non. POST impose que le serveur soit accessible et ainsi l'expose, mais les résultats sont reçus instantanément.

La notion de client et de serveur est floue, il ne peut pas être attribuée à un programme ou à un équipement, par exemple, le programme `display_server.py` est serveur pour Sigfox et client pour Beebotte. Le site de Sigfox est serveur, si l'on utilise une méthode GET pour obtenir les données et client s'il fait un POST.

Pour revenir au paradigme de REST, nous avons, avec CBOR, défini le contenu de la ressource mais nous ne l'avons pas nommée. Le récepteur doit connaître le format (du CBOR contenant une série temporelle codée en différentiel) et ce qu'elle désigne. C'est ce que nous allons voir dans la prochaine session avec le protocole **CoAP** où nous pourrons définir, comme en HTTP, le nom de la ressource et son contenu.

11. LoRaWAN

Avec **Sigfox**, ce n'était que du bonheur ! Nous avions un environnement unique développé par un seul acteur. L'attachement d'un objet au réseau, la récupération des données s'en trouve grandement simplifiée.

Avec **LoRaWAN**, l'écosystème est plus complexe. Pour rappel, **LoRa** est une modulation longue portée et LoRaWAN un protocole de niveau 2 qui a pour but de fédérer les acteurs (fabricants d'objets, utilisateurs d'objets, opérateur de réseaux) autour de la communication radio entre l'objet et le cœur de réseau appelé ici **LNS**. Mais l'enregistrement des objets, le lien entre les applications et le LNS diffèrent d'un réseau à un autre.



Même si vous n'avez pas l'intention de mettre en œuvre le réseau Sigfox, nous vous recommandons de lire le chapitre précédent (ou à la rigueur de le survoler) car nous y ferons référence dans la suite du texte.

LoRaWAN, comme Sigfox, opère dans la même bande de fréquences non licenciées. Il existe plusieurs opérateurs de réseaux LoRaWAN :

- The Things Network (TTN)¹ propose une approche communautaire. Chaque personne peut mettre à disposition des passerelles radio pour la communauté et inscrire des objets. TTN gère le cœur de réseau. Si cette approche est sympathique, la couverture va être très parcellaire et va dépendre de la densité de geeks dans une région. De plus, la couverture du réseau va dépendre de la position des antennes qui doivent être placées sur un point haut. Avoir une antenne sur son balcon n'implique pas une grande couverture de la zone. Si vous avez de la chance, vous pourrez peut-être bénéficier d'un accès via TTN. Nous verrons comment s'y connecter.
- des opérateurs nationaux comme, en France, **Orange** ou **Bouygues Télécom**. Mais il faut généralement bourse délier pour connecter les objets et pour limiter l'usage du downlink ; l'envoi des messages vers les capteurs est facturé en supplément. En revanche, la couverture

1. <https://www.thethingsnetwork.org/>

- est supérieure.
- les réseaux privés. Il est possible de faire tourner son propre LNS. Cela implique d'acheter des composants pour faire une passerelle radio. Mais il existe des mises en oeuvre ouverte du LNS comme **chirpstack**².

11.1 Information sur le LoPy

Chaque objet va avoir un identifiant unique codé sur 64 bits appelé ***devEUI***. Cette information est nécessaire pour faire entrer un équipement sur le réseau. Le programme `lorawan_devEUI.py` donne accès à cette valeur qui est unique pour chaque LoPy.

Listing 11.1 – `lorawan_devEUI.py`

```
from network import LoRa
2import pycom
import binascii
#
lora = LoRa(mode=LoRa.LORAWAN)
mac = lora.mac()
#
print ('devEUI:', binascii.hexlify(mac))
#
```

L'objet Python LoRa est importé du module `network` et une instance est créée ligne 5. La variable `mac` va stocker l'adresse MAC (autre nom du *devEUI*) retournée par l'objet `lora` et l'afficher en hexadécimal, comme le montre l'exemple suivant :

```
>>> Running lorawan_devEUI.py
>>>
>>>
devEUI: b'70b3d54994c61237'
```

On a le *devEUI*. Il manque encore 3 informations :

- ***appEUI*** : c'est un identifiant sur 64 bits comme le *devEUI* qui va servir à identifier l'application.
On peut le mettre à 0 ;
- ***AppKey*** : c'est une valeur sur 128 bits qui n'est connue que de l'objet et du LNS. Elle permet de dériver les clés de chiffrement utilisées par la suite ;

Et sur le LNS, il faudra aussi configurer le connecteur : il s'agit d'indiquer au LNS comment envoyer les données vers une application. Il peut s'agir de POST HTTP comme nous avons vu avec la configuration du ***backend*** avec Sigfox. Bien entendu, une fois ces grands principes établis, ça serait trop simple si tous les réseaux fonctionnaient de la même façon. Nous allons donc voir comment faire avec le réseau TTN.

11.2 The Things Network

2. <https://www.chirpstack.io/>

The Things Network, ou TTN pour les intimes, est un réseau communautaire qui permet à tout un chacun de connecter soit une passerelle radio pour le bénéfice de la communauté, soit un objet. TTN se charge de faire tourner un LNS et de renvoyer les informations collectées à son propriétaire. On espère qu'un bon samaritain a déployé une antenne près de chez vous pour capter votre trafic. On n'a aucun moyen de savoir, si on n'essaye pas.



La première étape consiste à se créer un compte, gratuit comme il se doit, sur le site de TTN : <https://www.thethingsnetwork.org/>.

Une fois le formulaire de création de compte rempli et validé, vous disposez d'un accès au réseau TTN. Votre identifiant apparaît en haut à droite de la page Web.

- Cliquez sur votre nom,
- sélectionnez Console,
- puis votre zone géographique (cela n'a rien à voir avec le plan de fréquence. Il est préférable de choisir le serveur le plus proche pour optimiser les communications).

On peut soit définir une application et y associer des objets, soit connecter une passerelle radio (*Go To Gateways*). Par la suite, nous explorerons cette option si vous voulez installer votre propre passerelle radio. Nous allons nous intéresser à la création d'application en choisissant *Go To Applications*. TTN vous invite à créer notre première application ; cliquez sur le lien *+Add Application*.

Définir une application

Une application pour TTN va intégrer plusieurs objets LoRaWAN qui enverront leur information à une application tournant sur un Application Server (AS). Il faut remplir le formulaire (cf. figure 11.1 page suivante) en indiquant :

- l'identifiant de l'application. Il s'agit d'un nom uniquement composé de lettres et de chiffres ainsi que du tiret. Il doit être unique pour TTN. Donc, faites preuve d'imagination car il doit être unique pour TTN. Ce nom se retrouvera dans les URI permettant de piloter l'application ;
- le nom de l'application qui apparaîtra dans les menus. Il n'y a pas de contraintes ; il vaut mieux choisir quelque chose d'explicite ;
- une description de l'application si nécessaire.

Cliquez sur *Create Application* pour enregistrer cette application. Vous arrivez sur une page de contrôle de votre application.

Ajout d'un objet

Nous devons enregistrer un ou plusieurs objets dans notre application :

- Cliquez sur *end devices* dans le menu de gauche ;
- puis *+ Add end device* ;
- et *try manual registration* en dessous du menu déroulant. TTN offre des aides à la configuration pour certains objets, mais comme nous sommes des experts, nous n'en n'avons pas besoin.

Le menu, décrit figure 11.2 page 135 apparaît :

- Entrez le plan fréquence compatible avec votre région. En Europe, vous pouvez choisir d'avoir

General settings

Application ID *

Name

Description

Attributes

+ Add attributes

Attributes can be used to set arbitrary information about the entity, to be used by scripts, or simply for your own organization

Skip payload encryption and decryption

Enabled

Skip decryption of uplink payloads and encryption of downlink payloads

FIGURE 11.1 – Configuration d'une application.

la voie descendante en SF9³ ou en SF12. Le premier est un bon compromis entre la portée et la durée d'émission ; le second est à choisir si votre objet est difficilement accessible (profondément enfoui ou loin d'une passerelle radio) ;

- la version du protocole LoRaWAN compatible avec les LoPy :MAC V1.0.2 ;
- les paramètres de la couche physique du LoPy : PHY V1.0.2 REV A ;
- Vous devez récupérer le *devEUI* de votre LoPY en lançant sur Atom le programme *lorawan_devEUI.py* (cf. Listing 11.1 page 132) ;
- mettez l'*AppEUI* à 0,
- et générez une clé de chiffrement *AppKey* et n'oubliez surtout pas de copier sa valeur quelque part, on en aura besoin pour configurer notre objet.

L'interface de TTN vous propose un identifiant pour l'objet basé sur son *devEUI*. Vous pouvez le garder, à moins de vouloir quelque chose de plus explicite.

Finally, click on *register end device* to save your work.

L'interface affiche une page récapitulative. Si vous avez oublié de le faire à l'étape précédente, copiez l'*AppKey* pour la mettre dans le programme *lorawan_send_and_receive.py*.

Connexion de l'objet

Listing 11.2 – *lorawan_send_and_receive.py*

```

from network import LoRa
import socket
import time
import pycom
import binascii

```

3. Le Spreading Factor définit la vitesse de transmission de l'information, elle diminue de moitié à chaque incrémentation donc le SF12 est 16 fois plus lent que le SF9, mais beaucoup plus robuste.

Register end device

From The LoRaWAN Device Repository [Manually](#)

Frequency plan ⓘ *

Select... | ▾

LoRaWAN version ⓘ *

Select... | ▾

Regional Parameters version ⓘ *

Select... | ▾

[Show advanced activation, LoRaWAN class and cluster settings](#) ▾

DevEUI ⓘ *

... | [Generate](#) 0/50 used

AppEUI ⓘ *

... | [Fill with zeros](#)

AppKey ⓘ *

... | [Generate](#)

End device ID ⓘ *

my-new-device

This value is automatically prefilled using the DevEUI

After registration

View registered end device

Register another end device of this type

FIGURE 11.2 – Ajout d'un objet.

```

6 lora = LoRa(mode=LoRa.LORAWAN, region=LoRa.EU868)
8 #
9 mac = lora.mac()
10 print('devEUI:', binascii.hexlify(mac))

```

Le programme `lorawan_send_and_receive.py` commence, comme `lorawan_devEUI.py`, par créer un objet `lora` et afficher le *devEUI* de l'objet, c'est toujours pratique pour le débogage.

```

12 # create an OTAA authentication parameters
13 app_eui = binascii.unhexlify(
14     '00u00u00u00u00u00u00'.replace('u', ''))
15
16 app_key = binascii.unhexlify(
17     '4EAE56D0689F6F8B02C2AFA7E08DADBA'.replace('u', ''))

```

On met dans deux variables, les identifiants présents sur le LNS de TTN, à savoir le *app_eui* que l'on avait mis à zéro et de *app_key* que l'on généré aléatoirement sur le site de TTN et que l'on vous avait demandé de copier quelque part.

Pour éviter de manipuler des séquences binaires, ces valeurs sont vues comme des chaînes de caractères (des espaces peuvent être insérés pour plus de lisibilité, la fonction `replace` permet de les éliminer). La séquence binaire est obtenue grâce à la fonction `unhexlify`.

```

20 pycom.heartbeat(False)
21 pycom.rgbled(0x101010) # white

```

On arrête le clignotement périodique bleu de la LED du LoPy en appelant la fonction `pycom.heartbeat` avec l'argument `False`. Puis on allume la LED en blanc avec la fonction `rgbled`. L'argument donne l'intensité des trois composantes Rouge, Vert et Bleu.

La LED va nous permettre de suivre pas à pas l'état du LoPy.

```

22 # join a network using OTAA (Over the Air Activation)
23 lora.join(activation=LoRa.OTAA, auth=(app_eui, app_key), timeout=0)
24
25 # wait until the module has joined the network
26 while not lora.has_joined():
27     time.sleep(2.5)
28     print('Not yet joined... ')
29
30 pycom.rgbled(0x000000) # black

```

Le LoPy effectue la connexion au réseau, cette procédure est appelée ***Join***. Elle correspond à l'émission d'un message vers le LNS. Si celui-ci reconnaît et accepte l'objet, il renverra quelques secondes plus tard un message *Accept*. Cette phase permet à l'Objet d'obtenir les clés de chiffrement des messages et une adresse plus courte appelé *devAddr*.

Pour se faire, le programme appelle la fonction `join` avec les paramètres suivants :

- le type de connexion, très souvent Over The Air Authentication (OTAA) pour récupérer dynamiquement les paramètres⁴,
- les paramètres nécessaires, à savoir l'*AppEUI* et l'*AppKey* que nous avons précédemment configurés.

4. Il existe aussi une méthode appelé Authentication By Personalisation (ABP) qui consiste à configurer l'objet avec tous ses paramètres (comme pour Sigfox), mais elle est moins souple et moins adaptée à l'environnement LoRaWAN où plusieurs opérateurs coexistent sur les mêmes fréquences.

- un temporisateur (*timeout*). La valeur à 0 indique que l’objet va essayer de joindre le réseau jusqu’à ce que celui-ci l’accepte.

La fonction *join* n’étant pas bloquante, la procédure de *join* va se dérouler en tâche de fond. La fonction *has_joined* permet de suivre l’état du LoPy. D'où la boucle (lignes 26 à 28) dont on ne sortira que lorsque le LoPy sera accepter sur le réseau.

Toutes de 2.5 seconde un message sera affiché pour dire qu'il est toujours en attente d'une réponse. Ces messages ne sont absolument pas synchronisés avec l'envoi des trames Join qui ont lieu toutes les 15 secondes.

L’extinction de la LED (ligne 30) indique que le LoPy a joint le réseau LoRaWAN.

```
32 s = socket.socket(socket.AF_LORA, socket.SOCK_RAW)
33 s.setsockopt(socket.SOL_LORA, socket.SO_DR, 5)
34 s.setsockopt(socket.SOL_LORA, socket.SO_CONFIRMED, False)
```

Le programme ouvre la socket, le premier paramètre est AF_LORA pour indiquer que l'on utilise le protocole LoRaWAN. Les deux lignes suivantes, avec les fonctions *setsockopt*, permettent de configurer des paramètres LoRaWAN, comme :

- le Data Rate (DR). Sous ce terme, plusieurs paramètres des la couche physique sont réunis. Plus le DR est élevé, plus la transmission est rapide. En contre partie, elle est moins robuste et peut porter moins loin. Le DR varie entre 0 et 6.
- la capacité de LoRaWAN a acquitter les trames. Il n'est pas recommandé de l'activer car cela induit des émissions qui sont prises en compte dans le calcul du **Duty Cycle** (voir chapitre 5.2 page 60).

```
36 while True:
37     pycom.rgbled(0x100000) # red
38     s.setblocking(True)
39     s.settimeout(10)
40
41     try:
42         s.send('Hello\LoRa')
43     except:
44         print ('timeout\in\sending')
45
46     pycom.rgbled(0x000010) # blue
```

On entre dans une boucle sans fin pour envoyer et recevoir périodiquement des messages :

- ligne 37, la LED éclaire en rouge pour indiquer une transmission.
- ligne 38, les appels aux sockets sont rendu bloquant, c'est-à-dire qu'il ne finiront que quand l'action sera effectuée par la couche inférieure.
- ligne 39, par contre au bout de 10 secondes d'attente une erreur sera déclenchée.
- ligne 41, on récupère cette erreur grâce aux instructions *try* et *except* de Python. Si l'appel à *send* reste bloqué plus des 10 secondes, le traitement de l'exception affiche *timeout in sending*. Cet événement est rare et est causé par une défaillance locale à la couche 2⁵.
- ligne 46 dans tous les cas, la LED passe en bleu.

```
50     data = s.recv(64)
```

5. Il faut rester vigilant sur cette erreur, par exemple si l'on essaie d'envoyer un entier sans sérialisation, cela conduira à une erreur qui sera prise en compte de cette manière.

```

52     print(data)
53     pycom.rgbled(0x001000) # green
54 except:
55     print('timeout in receive')
56     pycom.rgbled(0x000000) # black
57
58 s.setblocking(False)
59 time.sleep(29)

```

Finalement, on attend des données. Dans le mode le plus fréquent de LoRaWAN, un équipement ne peut recevoir des données que dans une courte fenêtre après une émission. Si des données arrivent en dehors de cette fenêtre, le LNS mémorise l'information et la transmettra quand il recevra une trame de l'objet. Cela permet d'économiser beaucoup d'énergie car le capteur peut dormir de ses deux oreilles, il sait qu'il ne manquera pas une information.

La voie descendante est une ressource rare, et il ne faut généralement pas trop l'utiliser⁶, donc il ne devrait pas y avoir souvent de messages en retour. Sauf ici, pour tester toute la chaîne de transmission :

- ligne 49, l'appel à `recv` est aussi bloquant tant que des données ne sont pas arrivées.
- lignes 51 et 52, si des données arrivent, elles sont affichées et la LED passe au vert.
- lignes 53 à 55, si aucune données arrivent, le temporisateur se déclenche au bout de 10 secondes générant l'exception. Un message averti l'utilisateur de l'absence de données et la LED est éteinte.

Si tout va bien, qu'une passerelle radio est à portée de votre objet, le programme `lorawan_send_and_receive.py` va se connecter au réseau après avoir affiché 3 fois `Not yet joined...` :

```

>>> Running lorawan_send_and_receive.py
>>>
>>>
devEUI: b'70b3d54994c61237'
Not yet joined...
Not yet joined...
Not yet joined...
timeout in receive
timeout in receive

```

correspondant au 5 secondes prévues par le standard avant d'envoyer le message d'acceptation de l'objet.

La LED passe du blanc au rouge indiquant qu'un message a été émis puis doit s'éteindre et le message `timeout in receive` est affiché indiquant que l'Objet n'a pas reçu de message en réponse à son émission.

Analyse des traces

Côté LNS, il est possible de voir le trafic sur la page résumant les caractéristiques de l'objet dans la fenêtre *live data* (cf. figure 11.3 page suivante).

⁶. TTN ne facture pas les données descendantes, mais d'autres opérateurs le font, faites attention si vous adaptez ces exemples à d'autres environnements, car dans les premiers tests, nous utiliseront la voie de retour.

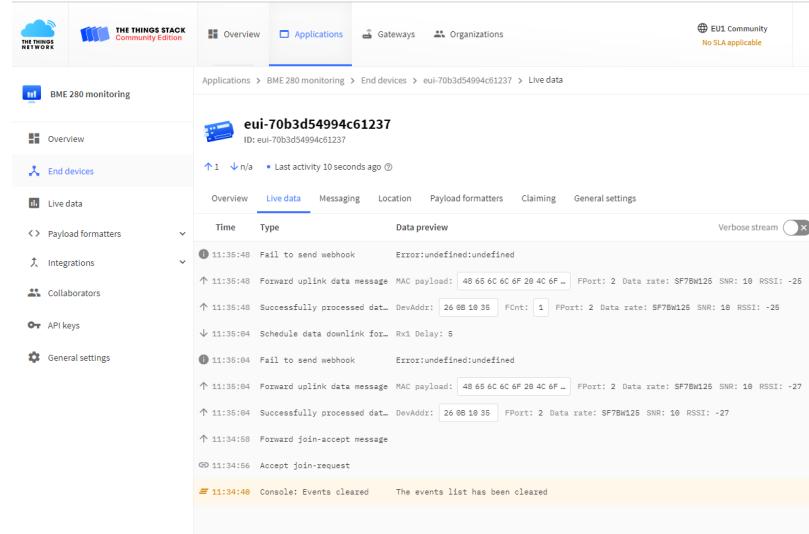


FIGURE 11.3 – Trace d'un objet.

Les événements sur la figure se lisent de bas en haut comme le confirme l'horodatage, en cliquant sur la ligne correspondante, TTN affiche les messages JSON utilisés par le LNS pour traiter l'information. On y retrouve les données échangées :

- L'événement *Accept Join-request* indique que le LNS a reçu le message Join de l'objet, indiquant qu'il voulait rejoindre le réseau. Le LNS l'a traité, a vérifié qu'il y avait un objet déclaré avec le bon ***devEUI*** et que l'***AppKey*** utilisée par l'objet pour signer sa demande était identique à la valeur configurée. Dans la trace suivante, on peut retrouver le ***devEUI*** de l'objet et la ***devAddr***, qui lui a été attribuée temporairement. Cette dernière sera utilisée par la suite.

```
"device_ids": {
    "device_id": "eui-70b3d54994c61237",
    "application_ids": {
        "application_id": "plido-appl"
    },
    "dev_eui": "70B3D54994C61237",
    "join_eui": "0000000000000000",
    "dev_addr": "260B1035"
}
```

- L'événement *Forward Join-accept message* indique que le LNS envoie un message d'acceptation à l'Objet. Ce message est différé de 5 secondes pendant lesquelles l'Objet dort.
- L'événement *Successfully processed data message* montre que le LNS a reçu correctement le message de données émis par l'Objet. La trace donne des indications sur certains paramètres physique, tels que le Spreading Factor (ici SF7 soit la modulation la plus rapide et la moins robuste), le SNR qui indique le ratio entre la puissance du signal reçu par rapport au bruit et le Received Signal Strength Indicator (RSSI) donnant la force du signal.
- L'événement *Forward uplink data message* donne plus d'information sur le message et son contenu. Le champs **frm_payload** donne le contenu déchiffré par le LNS mais codé en **base64**. On y retrouve le message d'origine Hello LoRa.

```
"uplink_message": {
```

```

    "session_key_id": "AX4VN0uMbClcPKF63sKy6A==" ,
    "f_port": 2,
    "frm_payload": "SGVsbG8gTG9SYQ==",
    "rx_metadata": [
        {
            "gateway_ids": {
                "gateway_id": "gateway-lo",
                "eui": "B827EBFFFE08F8A8"
            },
            "time": "2022-01-01T10:35:04.247807Z",
            "timestamp": 2661650827,
            "rssi": -27,
            "channel_rssi": -27,
            "snr": 10,
            "uplink_token": "ChgKFgoKZ2FOZX....",
            "channel_index": 1
        }
    ]
}

```

Question 11.2.1: Mauvais appKey

Quel message de trace obtenez vous du LNS, si l'Objet n'est pas configuré avec le même **AppKey** que le LNS ?

Question 11.2.2: Autonomie

Quel impact sur les batteries ?

Le **fPort** que l'on retrouve dans les trames de données LoRaWAN permet d'indiquer le programme qui va traiter l'information. La valeur 0 est spéciale et indique une trame de configuration de la couche LoRaWAN, les autres valeurs jusqu'à 223 désignent des applications spécifiques. Bien qu'il soit peu utilisé, il est parfois utile de la changer.

Question 11.2.3: fPort

Dans les traces précédentes, quelle est la valeur du fPort par défaut utilisé par le LoPy ?

Question 11.2.4: Changement de fPort

L'instruction bind permet de modifier le **fPort** pour une transmission. Modifiez le programme pour utiliser le fPort 10 et vérifiez l'effet dans les traces de TTN.

Configuration du connecteur

Le LNS n'a pas été configuré pour envoyer le message à un Serveur d'Application (Application Server). Comme pour **Sigfox**, il faut définir une URI pour indiquer au LNS où poster les données.

Pour configurer un connecteur, il faut aller sur le menu de gauche *Integrations*. TTN propose des intégrations directes avec des services cloud existants, mais il est également possible de configurer son propre connecteur :

- via **MQTT**, TTN joue le rôle de *broker* et fourni le nom du **topic** et le secret.
- en stockant localement des données par le choix **Storage integration** et en y accédant par une requête GET, comme Sigfox le propose (cf. chapitre 10.5.3 page 124). Cela permet de recevoir les données si l'on est situé derrière un NAT, mais rend la réception asynchrone puisque l'application doit interroger régulièrement le serveur de TTN.
- en configuration une URI sur le serveur pour que celui-ci produit une requête POST quand une trame de donnée est reçue (choix **Webhooks**).

Nous allons également privilégier pour TTN ce dernier mode de fonctionnement, car si il nécessite une configuration du NAT, il permet de recevoir les données de manière synchrone lorsqu'elles sont traitées par le LNS.

Il faut cliquer sur **Webhooks**. Un certain nombre de services sont proposés, mais nous allons définir le nôtre à l'instar de ce que nous avons fait avec Sigfox (cf. chapitre 10.5.4 page 126). Pour préparer la connectivité, il est nécessaire :

- Si vous êtes dans le cloud, l'adresse s'obtient en tapant la commande `ifconfig`.
- Si vous êtes derrière une box ou le routeur de votre fournisseur d'accès, vous avez une adresse privée. Un site web comme `myip.wtf` vous retournera l'adresse IPv4 publique. Vous devez ensuite configurer votre router d'accès pour qu'il envoie les paquets TCP ayant le numéro de port 9999 à votre ordinateur.

Cela va permettre de former l'URI suivante :

```
http://aaa.bbb.ccc.ddd:9999/ttn
```

où `aaa.bbb.ccc.ddd` représente l'adresse IP publique de votre équipement. Comme cette adresse peut changer au cours du temps, il est préférable d'utiliser un service de DNS dynamique pour obtenir un nom de domaine plus stable sur la durée.

La page listant les **Webhooks** associés à l'application est vide. Il faut cliquer sur `pour en ajouter un nouveau`, il convient de choisir *Custom webhook*, les autres définissent les connecteurs vers des services de gestion des données (cf. figure 11.4 page suivante).

Dans ce menu, il faut indiquer :

- l'identifiant du webhook :
- le format JSON :
- l'URI que nous venons de construire :
- il n'est pas nécessaire pour l'instant de remplir la *Download API Key* qui servira pour envoyer des données à l'Objet. Nous y reviendrons par la suite.
- il faut choisir les événements qui déclencheront une requête POST. Dans notre cas, on ne choisit que la réception d'un message montant (*Uplink message*). On peut compléter le chemin de l'URI précédemment définie avec des éléments qui permettront au serveur REST de mieux identifier le type de requête. Dans notre cas, cela n'est pas nécessaire. `/ttn` identifie par conséquent les messages reçus par le LNS des Objets.

Une fois le *webhook* validé, le programme `generic_relay.py` doit être lancé. Il va créer un serveur Web qui va attendre les requêtes POST du LNS de TTN.

```
94 @app.route('/ttn', methods=['POST']) # API V3 current
def get_from_ttn():
    fromGW = request.get_json(force=True)
```

Applications > BME 280 monitoring > Webhooks > Add > Custom webhook

Add webhook

General settings

Webhook ID *
plido-webhook

Webhook format *
JSON

Endpoint settings

Base URL
http://:9999/ttn

Downlink API key

The API key will be provided to the endpoint using the "X-Downlink-Apikey" header

Additional headers
+ Add header entry

Enabled messages

For each enabled message type, an optional path can be defined which will be appended to the base URL

Uplink message
 Enabled /path/to/webhook

Join accept
 Enabled

FIGURE 11.4 – Configuration d'un webhook.

```

96     downlink = None
98     if "uplink_message" in fromGW:
100         payload = base64.b64decode(fromGW["uplink_message"]["frm_payload"])
102         downlink = forward_data(payload)
104         if downlink != None:
105             from ttn_config import TTN_Downlink_Key
106
107             downlink_msg = {
108                 "downlinks": [
109                     {
110                         "f_port": fromGW["uplink_message"]["f_port"],
111                         "frm_payload": base64.b64encode(downlink).decode()
112                     }
113                 ]
114             }
115             downlink_url = \
116                 "https://eu1.cloud.thethings.network/api/v3/as/applications/" + \
117                 fromGW["end_device_ids"]["application_ids"]["application_id"] + \
118                     "/devices/" + \
119                     fromGW["end_device_ids"]["device_id"] + \
120                     "/down/push"
121
122             headers = {
123                 'Content-Type': 'application/json',
124                 'Authorization' : 'Bearer' + TTN_Downlink_Key
125             }
126
127             x = requests.post(downlink_url,
128                               data = json.dumps(downlink_msg),
129                               headers=headers)
130
131             resp = Response(status=200)
132             return resp
133
134
135
136
137
138

```

Ce programme utilise le module **Flask** pour créer un serveur Web. Il associe le chemin /ttn à la fonction `get_from_ttn` pour les requêtes de type POST (lignes 93 et 94). Quand une requête de ce type arrive, la variable `fromGW` contient l'objet JSON⁷ (ligne 95). Si elle contient la clé `uplink_message`, les données sont extraites (ligne 100) pour être envoyé au travers de l'interface `loopback` grâce à la fonction `forward_data` à un programme qui les traite. Si cette fonction retourne la valeur `None`, le programme ne souhaite pas répondre à l'Objet, le serveur Web acquitte positivement (code 200) la requête venant du LNS (ligne 130 et 131).

```

def forward_data(payload):
    global verbose, forward_port, forward_address

    inputs = [sock]
    outputs = []

    if verbose:
        print ("--UP->", binascii.hexlify(payload))
        sock.sendto(payload, (forward_address, forward_port))

    readable, writable, exceptional = select.select(inputs, outputs, inputs, 0.1)

```

7. convertie implicitement en dictionnaire Python.

```

40     if readable == []:
41         if verbose:
42             print ("no DW")
43         return None
44
45     for s in readable:
46         replyStr = s.recv(1000)
47         if verbose:
48             print ("<-DW--", binascii.hexlify(replyStr))
49         return replyStr
50
51     return None

```

Si le mode verbose est activé, le message à envoyer au programme de traitement des données est affiché en hexadécimal grâce à la fonction `hexlify` (lignes 33 et 34). Puis le message est effectivement émis (ligne 35)⁸

Pour attendre une réponse qui n'est pas systématique, on ne peut pas utiliser l'appel `recvfrom` car celui-ci est bloquant. A la place, on utilise l'appel `select` du module du même nom. Il permet d'attendre sur plusieurs événements à la fois provenant de différentes sockets (ligne 37) où :

- `inputs` est un tableau des sockets pouvant avoir reçu des données. Dans notre cas (ligne 30) il est initialisé avec la socket dialoguant avec le programme traitant les données ;
- `outputs` est un tableau qui contient les sockets dans lesquelles il serait possible d'écrire. Dans notre cas, ce tableau est vide (ligne 31).
- le troisième concerne les exceptions. Dans notre cas, on répète le tableau des sockets `inputs`
- finalement le quatrième paramètre indique la durée d'attente dans la fonction `select`. Dans notre cas, il est fixé à 0.1 seconde. ce temps est à la fois compatible avec une réponse du programme de traitement des données, et la réponse que l'on doit envoyer au LNS pour que la réponse soit associée au message montant.

`select` retourne dans un tuple de trois éléments, la ou les sockets qui ont déclenché son retour.

Dans notre cas, se sont :

- soit l'expiration du temporisateur, dans ce cas, la variable `readable` qui contient la liste des sockets ayant reçues des données est vide (ligne 39) et la valeur `None` est renournée.
- soit des données qui sont arrivées dans une ou plusieurs socket. la variable `readable` contient le tableau de ces sockets. La variable `s` contient la première socket du tableau, les données sont lues et envoyées en retour.

Le programme `generic_relay.py`, lancé avec l'option `-v` montre de manière synthétique le trafic.

```

>python3 generic_relay.py -v
--UP-> b'48656c6c6f204c6f5261'
no DW
63.34.43.96 - - [01/Jan/2022 18:27:45] "POST /ttn HTTP/1.1" 200 -
--UP-> b'48656c6c6f204c6f5261'
no DW
34.255.49.188 - - [01/Jan/2022 18:28:31] "POST /ttn HTTP/1.1" 200 -

```

Le contenu du message montant est affiché en hexadécimal et correspond au fameux contenu Hello LoRa. Comme il n'y a pas de réponse, no DW est ensuite affiché.

8. l'adresse du destinataire et le port sont configurés lors de l'analyse des paramètres.

Traitement des données

La fonction `forward_data` du programme `generic_relay.py` envoie les données à l'adresse de **loopback** 127.0.0.1 et sur le port 33033⁹.

Le programme `display_receive_and_send.py` traite les données de l'Objet.

Listing 11.3 – `display_receive_and_send.py`

```

1 import socket
2 import binascii
3
4 s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
5 s.bind(('0.0.0.0', 33033))
6
7 while True:
8     data, addr = s.recvfrom(1500)
9     print (data)
10    s.sendto("Pleased to meet you!".encode(), addr)
```

Il attend les données sur le port 33033. La fonction `recvfrom` retourne les données stockées dans la variable `data` et l'adresse de celui qui a envoyé les données stockée dans la variable `addr`. Les données sont affichée ligne 9 et le message `Pleased to meet you` est retourné à l'émetteur, en ligne 10 grâce à la fonction `sendto`.

En lançant le programme dans une nouvelle fenêtre, on reçoit correctement le message du LoPy.

```
>python3 display_receive_and_send.py
b'Hello LoRa'
```

En revanche, le programme `generic_relay.py` produit une erreur.

```
--UP-- b'48656c6c6f204c6f5261'
<-DW-- b'506c656173656420746f206d65657420796f7521'
63.34.43.96 - - [01/Jan/2022 19:06:51] "POST /ttn HTTP/1.1" 500 -
Traceback (most recent call last):
... 
```

Le message en retour a été reçu par la fonction `forward_data`, mais la voie descendante n'a pas encore été configurée. L'erreur est liée à l'absence du fichier `ttn_config.py`, ligne 104, ce qui force Flask à retourner le status 500 et afficher les fonctions qui ont conduit à l'erreur.

Voie retour

Il n'y a aucun problème de sécurité quand le LNS envoie au serveur d'application des données reçues de capteurs, car l'endroit où il les transmet a été configuré par leur propriétaire. Par contre, dans l'autre sens, le LNS doit s'assurer que les données à transmettre à l'Objet, proviennent bien d'un serveur d'application autorisé.

Comme pour Beebotte, une clé secrète va être utilisée. A cette clé, le LNS peut associer des droits pour plus ou moins limiter les accès. Dans le menu de gauche, cliquez sur *API key* puis sur *+ Add API Key*. Dans l'écran suivant, donnez un nom à votre clé et choisissez *Grant Individual Right* pour limiter l'usage de la clé. Sélectionnez *Write downlink application traffic* et validez.

⁹. Ces deux valeurs peuvent être respectivement changées en utilisant les arguments `--forward_address` et `--forward_port`.

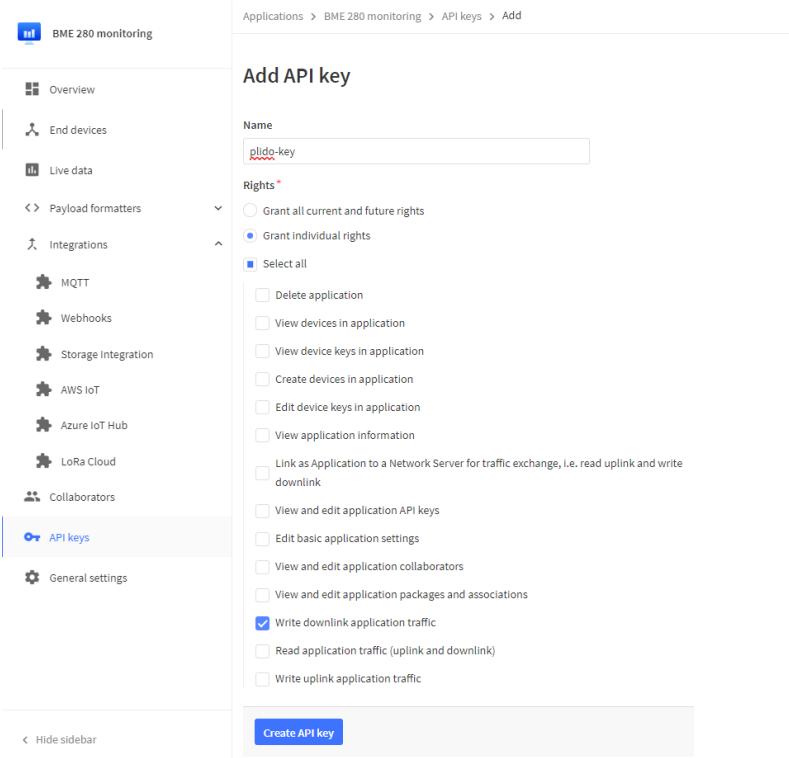


FIGURE 11.5 – Création d'une clé

Recopiez la clé et confirmer que vous l'avez bien fait en validant *I have copied the key*. Editez le fichier `ttn_config.py` en y insérant la clé obtenue.

Listing 11.4 – `ttn_config.py`

```
1 TTN_Downlink_Key = "ENTER YOUR KEY"
```

Relancez le programme `generic_relay.py`, le LoPy affiche la réponse.

Le listing suivant reprend la construction de la requête POST vers le LNS pour transporter le message descendant :

```
104     if downlink != None:
105         from ttn_config import TTN_Downlink_Key
106
107         downlink_msg = {
108             "downlinks": [
109                 "f_port": fromGW["uplink_message"]["f_port"],
110                 "frm_payload": base64.b64encode(downlink).decode()
111             ]
112         }
113         downlink_url = \
114             "https://eu1.cloud.thethings.network/api/v3/as/applications/" + \
115             fromGW["end_device_ids"]["application_ids"]["application_id"] + \
116                 "/devices/" + \
117                 fromGW["end_device_ids"]["device_id"] + \
118                 "/down/push"
```

```

118     headers = {
119         'Content-Type': 'application/json',
120         'Authorization' : 'Bearer' + TTN_Downlink_Key
121     }
122
123     x = requests.post(downlink_url,
124                         data = json.dumps(downlink_msg),
125                         headers=headers)

```

- ligne 104, la clé autorisant l'envoi de message descendant est copié dans la variable TTN_Downlink_Key.
- lignes 106 à 110, le format JSON attendu par le LNS pour envoyer un message descendant est construit, plusieurs messages peuvent être envoyés à la fois, d'où l'utilisation d'un tableau. Chacun des éléments contient un fPort qui est recopié du message montant et des données qui proviennent de la socket (variable downlink codé en base64).

```
{'downlinks': [f'frm_payload': 'UGx1YXNlZCB0byBtZWV0IH1vdSE=', 'f_port': 10]}]
```

- lignes 111 à 116, l'URI pour accéder au service est construite. Elle contient deux éléments variables qui identifient l'application et l'Objet. Ceux-ci sont également extraits du message montant.
- lignes 118 à 121, les options qui seront ajoutées à l'en-tête HTTP sont indiquées. Elles contiennent la clé d'authentification.
- finalement, ligne 123 à 125, ces différents éléments sont passés à la fonction post du module requests.

11.3 Ajout d'une passerelle radio

Vous voulez installer une passerelle radio LoRaWAN chez vous pour bénéficier d'un réseau longue portée. Nous avons choisi d'utiliser un **Pygate** qui nous permettra de rester dans l'univers Pycom que nous connaissons bien maintenant et surtout qui est une des solutions les moins chères.

11.3.1 Installation du Pygate

Pour cela vous devez posséder une carte spécifique qui agit comme une carte d'extension. On y connecte un processeur. Ce dernier n'est pas forcément un LoPy car nous n'utiliserons pas sa partie radio LoRa. En effet, le Pygate dispose d'un autre composant radio Semtech qui permet l"écoute simultanée de huit canaux LoRa. Un LoPy lui ne peut écouter que sur un seul canal à la fois. En effet lors de l'émission d'une donnée, l'objet choisi aléatoirement un canal radio pour que le message soit reçu. Il faut que la passerelle radio écoute sur tous les canaux possibles.

On insère le Pycom sur la Pygate et on connecte l'antenne sur le connecteur du Pygate. Dans un second temps on doit adapter le Pycom à ses nouvelles fonctions, en téléchargeant le bon microprogramme. On le fait en lançant le programme Pycom Firmware Update et en choisissant Pygate.

Youtube



Dans un troisième temps, on ouvre le répertoire Pygate avec Atom. On y trouve le fichier `wifi_conf.py` qui va permettre à la Gateway d'avoir accès à votre réseau wifi, il peut contenir les mêmes identifiants que lorsque vous aviez connecter votre LoPY à votre réseau wifi. Le programme `main.py` est préconfigurée pour utiliser le LNS européen de The Things Network.

Le fichier `config.json` contient la description des fréquences utilisables dans cette zone. En y jetant un coup d'œil, on peut voir huit fréquences qui sont écoutées par la passerelle radio. Vous trouverez sur le site de The Things Network les autres configurations possibles.

On téléverse le programme dans la mémoire du Pycom de la passerelle radio et comme il s'appelle `main.py` il va se lancer tout seul au redémarrage. Il se connecte au wifi, puis va afficher l'identifiant la passerelle. Recopiez le car nous allons avoir besoin plus tard. Après on a plein de texte indiquant que la passerelle est en train de configurer le réseau LoRaWAN. Quand la led passe au vert, la passerelle est active.

11.3.2 Configuration de The Things Network

Maintenant nous devons informer The Things Networks de l'existence de notre passerelle radio. Si ce n'est pas le cas vous devrez créer un compte sur The Things Network. Aller sur console choisissez *Europe 1* si vous êtes en Europe ou la zone géographique correspondante. Ajoutez une passerelle en donnant :

- un identifiant textuel qui est un non lisible sans espaces ;
- l'identifiant de la passerelle qui correspond à la valeur hexadécimale affiché au démarrage ;
- un texte pour mieux décrire la passerelle ;
- finalement le plan fréquences¹⁰.

On valide la passerelle et elle est ajoutée au LNS. Vous allez pouvoir visualiser les trames qu'elle reçoit.

11.4 Vue générale des échanges

La figure 11.6 page suivante en illustre le fonctionnement global. La première émission du LoPy est captée par un ou plusieurs passerelles radio, qui relaient le message via un format JSON spécifique appelé **Packet Forwarder**, cela peut être de directement en UDP ou en utilisant MQTT. Le LNS après identification de l'Objet, envoie une requête POST contenant les données émise vers une destination configurée par une URI. Cette requête arrive au routeur du domicile, qui l'envoie à un équipement dans le domicile en fonction de la configuration du NAT. Le programme `generic_relay.py` traite la requête, en extrait le contenu et l'envoie via une socket au programme. Si ce dernier ne répond pas, le temporisateur se déclenche et la requête est acquittée.

Sinon, si le programme `generic_relay.py` reçoit des données en réponse avant le déclenchement du temporisateur, il initie une nouvelle requête POST vers le LNS. Dans notre cote, quand le LNS l'acquitte, la requête initiale est à son tour acquittée.

10. si les objets sont facilement accessibles on peut choisir le Spreading Factor 9 pour la voie descendante sinon si vous n'arrivez pas à joindre vos objets, vous pouvez vous replier sur le Spreading Factor 12

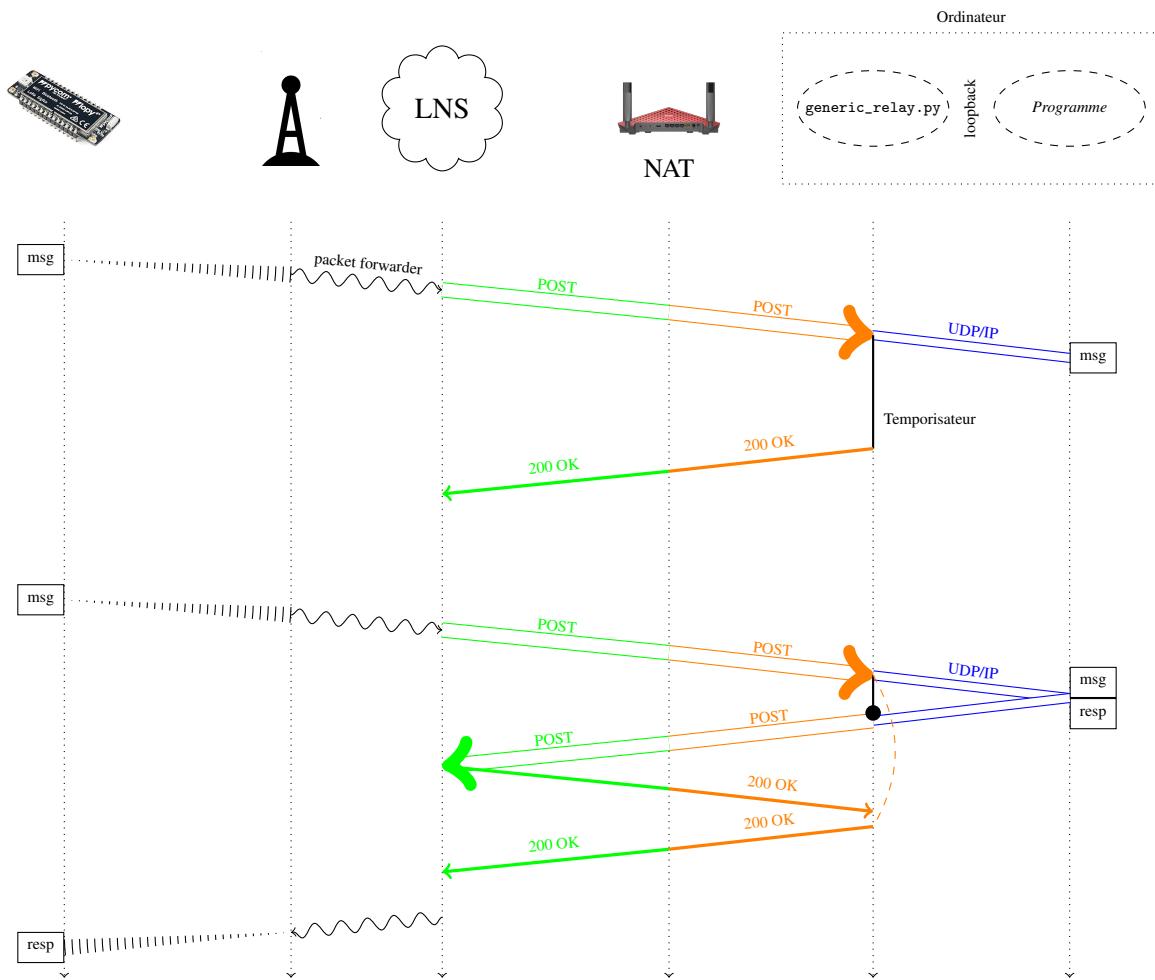


FIGURE 11.6 – Diagramme temporel des échanges.

Le message descendant est gardé en mémoire dans le LNS jusqu'à ce que la fenêtre de réception du Lopy s'ouvre. The downlinked message is held in memory in the LNS until the Lopy receive window opens.

11.5 Thermomètre LoRaWAN

Le programme `lorawan_temperature.py` combine la manière de rejoindre le réseau LoRaWAN que l'on a vu précédemment et la partie envoie des séries temporelles différentielles des mesures de température. La seule difficulté réside dans la taille de cette série temporelle. En effet, suivant le **Data Rate** le volume de données transportées diffère. Le tableau 12.2 page 156 donne la correspondance entre le *Data Rate* et les autres paramètres physiques. Il indique également la taille maximales des données transportées.

Ce changement de taille peut poser des problèmes de compatibilités, si l'on choisit une taille trop grande pour être transmise. En théorie, l'on connaît le *Data Rate* choisi et on peut adapter la taille en conséquence, mais l'opérateur peut aussi modifier le *Data Rate* de l'Objet en fonction

Data Rate	Spreading Factor	Largeur de bande	Taille Maximum (octets)
DR0	SF12	125 KHz	51
DR1	SF11	125 KHz	51
DR2	SF10	125 KHz	51
DR3	SF9	125 KHz	115
DR4	SF8	125 KHz	242
DR5	SF7	125 KHz	242
DR6	SF7	250 KHz	242

TABLE 11.1 – Data Rate en Europe

des conditions de transmission, et il n'existe aucune instruction en micro-python pour récupérer le *Data Rate* réellement utilisé. Pour simplifier la mise en œuvre, la taille maximale de 50 octets a été choisie.

Il suffit de remplacer `display_receive_and_send.py` par `display_server.py` pour que la série temporelle soit traitée et le résultat envoyé à Beebotte.

Question 11.5.1: US902

En vous aidant du lien suivant <https://www.thethingsnetwork.org/docs/lorawan/regional-parameters/> indiquant les paramètres régionaux. Est ce que le programme `lorawan_temperature.py` fonctionnerait sur un réseau LoRaWAN situé en Amérique du Nord ?



12. CoAP

Les principes REST avec leur représentation de l'information par des ressources pointées par des identifiants globalement unique est une des clés du succès de l'Internet et de la composition de services distribués. Même si ce n'est pas l'unique solution, il est indispensable que les informations provenant d'objets contraints puissent s'intégrer dans cette toile d'araignée mondiale. CoAP, pour Constrained Application Protocol, permet cette intégration à un meilleur coût en termes d'empreinte mémoire ou protocolaire que ne le permettrait le protocole HTTP.

12.1 Introduction

Dans les chapitres précédents, nous avons vu que CBOR permettait d'envoyer des données structurées de manière efficace, que le récepteur pouvait faire la différence entre un entier ou une chaîne de caractères et également savoir combien d'éléments comptaient un dictionnaire ou un tableau. En plus d'un gain de place par rapport à JSON, la complexité pour sérialiser ou déserialiser était limitée, conduisant à des implémentations peu gourmandes en mémoire.

Mais CBOR n'est pas suffisant pour une bonne interopérabilité. Quand un récepteur reçoit les données, il faut qu'il sache qu'il s'agit d'un codage CBOR et pas d'une autre structure. De plus, il faut que le récepteur sache quoi faire de ces données. Dans les exercices, nous avons transmis des séries temporelles correspondant à des relevés de température. Mais si nous voulions également transmettre l'humidité et la pression, comment le récepteur ferait la différence ?

Nous avons une solution pour répondre à ces questions : l'utilisation de ressources.

Nous avons vu qu'avec le paradigme REST, les ressources étaient nommées. Donc, pour distinguer les différentes séries temporelles, il suffit d'utiliser un nom (ou un URI) différent. Les ressources contiennent également des méta-informations et il est donc possible de transporter le format de codage pour indiquer qu'il s'agit de CBOR, de JSON, de CSV...

Malgré son universalité, ce modèle pose un problème pour les objets contraints :

Ver	T	TKL	code	Message ID
Token (if any)				
Options (if any)				
11111111		Payload (if any)		

FIGURE 12.1 – Format d'un message CoAP

HTTP utilise TCP pour fiabiliser les communications entre le client et le serveur. Or, TCP est gourmand en ressources. Il faut de la mémoire pour stocker les paquets non acquittés ou hors séquence, un grand nombre d'heuristiques doivent être mises en œuvre pour améliorer ses performances ; la souplesse pour créer des en-têtes peut s'avérer être un désavantage pour un objet constraint. Ainsi, la requête HTTP vers le serveur `www.arduino.cc` peut avoir cette forme :

```
GET / HTTP/1.1\r\n
Host: www.arduino.cc\r\n
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:25.0) Gecko/20100101 Firefox/25.0\r\n
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\r\n
Accept-Language: en-US,en;q=0.5\r\n
Accept-Encoding: gzip, deflate\r\n
Connection: keep-alive\r\n
\r\n
```

En dessous de la première ligne qui demande la ressource à la racine (généralement `index.html`), on trouve un certain nombre de lignes sous le format :

Nom du champ : valeur du champ*

qui vont indiquer au serveur le nom du serveur que l'on veut atteindre, la description du navigateur et les formats que celui-ci peut accepter.

Le but de CoAP est de définir un protocole beaucoup plus strict qui sera donc plus facile à mettre en œuvre mais qui pourra interopérer avec HTTP afin de préserver les principes définis par l'architecture REST et profiter du nommage des ressources pour que les ressources contraintes participent à la grande toile d'araignée mondiale.

12.2 Format d'une en-tête CoAP

L'en-tête des messages CoAP est de taille variable mais très structurée, comme le montre la figure 12.1.

Le premier mot de 32 bits est présent dans tous les messages CoAP :

- le champ Ver, sur 2 bits, contient le numéro de version du protocole, qui vaut 01 dans la version actuelle ;
- le champ T pour **Type**, également sur 2 bits, indique la nature du message (00 : **C**ONfirmable, 01 : **N**ON confirmable, 10 : Acquittement, 11 : Reset) ;
- le champ TKL, sur 4 bits, donne la longueur en octets du champ token démarrant au deuxième mot de 32 bits. Si la valeur est 0, ce champ est absent. Les valeurs de 1 à 8 indiquent la longueur. Les valeurs de 9 à 15 ne sont pas autorisées ;

Youtube



- le champ **Code**, sur 1 octet, permet un codage assez subtil de la nature de la requête ou de la réponse (cf. chapitre suivant) ;
- le champ **Message ID**, sur 2 octets, identifie les requêtes. the Message ID field, on 2 bytes, identifies the requests.

12.2.1 Codage de code

Dans beaucoup de protocoles applicatifs comme FTP ou HTTP, le serveur renvoie un code sur 3 caractères indiquant si la requête s'est exécutée correctement ou non. Les codes commençant par le chiffre :

- 1 informent que la requête est en train d'être traitée normalement. Ce type de notification n'est pas pris en compte avec CoAP ;
- 2 indiquent que la requête a été acceptée et traitée correctement ;
- 3 permettent d'indiquer une indirection ;
- 4 font référence à une erreur du côté client, due à une mauvaise syntaxe ou une requête qui ne peut être traitée. Ainsi la célèbre erreur 404 indique que le client a demandé une page qui n'existe pas sur le serveur ;
- 5 désignent une erreur du côté du serveur.

Le site web de l'Internet Assigned Numbers Authority (IANA)¹ donne les erreurs que l'on retrouve dans le protocole HTTP. Comme indiqué précédemment, le chiffre de gauche varie entre 1 et 5 tandis que les deux chiffres de droite, précisant la raison de la notification, varient généralement entre 0 et 31.

Pour permettre une représentation plus compacte, CoAP va coder cette chaîne de caractères dans un octet. Les trois bits de gauche désignent la nature du code et les 5 à droite donneront la raison.

Ainsi le code d'erreur HTTP 415 (*Unsupported Media Type*) se note en CoAP 4.15, s'écrit en binaire 100.01111 et en décimal 143. Cette notation concerne les réponses aux requêtes mais elle laisse de la place pour coder également les requêtes. En effet, le code avec les trois premiers bits à 0 n'est pas utilisé pour coder les notifications.

Plusieurs requêtes compatibles avec l'architecture REST peuvent être codées :

- **GET**, codé 0x01, retrouve le contenu d'une ressource présente sur le serveur et désignée par un URI ;
- **POST**, codé 0x02, stocke une valeur sur une ressource existante présente sur le serveur ;
- **PUT**, codé 0x03, crée une ressource sur le serveur et lui affecte une valeur ;
- **DELETE**, codé 0x04, supprime une ressource sur le serveur.

Notez que la valeur 0x00 peut être utilisée dans certains cas.

12.2.2 Utilisation du champ Message ID

Le champ **Message ID** sur 2 octets sert à identifier les messages CoAP afin de détecter les duplicitas. Cette valeur est recopiée dans les acquittements pour permettre de savoir quel message est acquitté. Ils ne doivent pas être réutilisés pendant une période fixée.

1. <http://www.iana.org/assignments/http-status-codes/http-status-codes.xhtml>
<http://www.iana.org/assignments/http-status-codes-1>



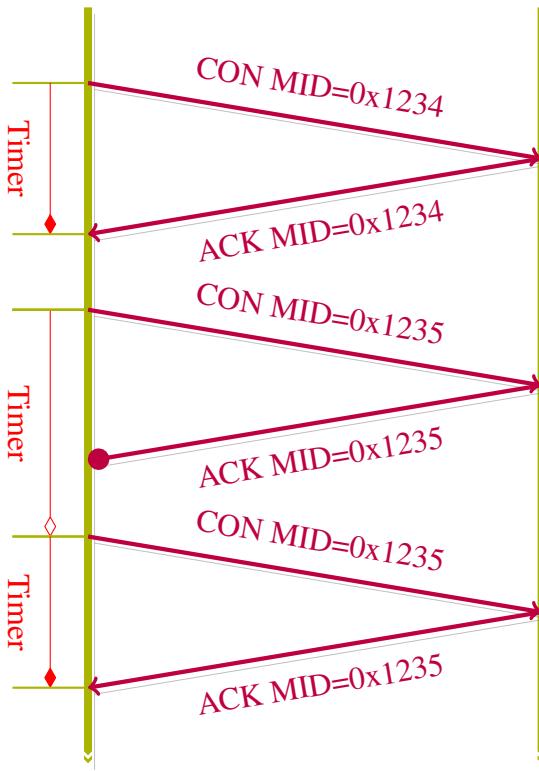


FIGURE 12.2 – Échanges fiabilisés avec CoAP

Le protocole CoAP repose sur la couche UDP pour des raisons de simplicité de mise en œuvre. Il peut être parfois nécessaire de fiabiliser le transfert des données. Pour se faire, CoAP dispose d'une sous-couche implantant un protocole très simple. Chaque message contient un champ `message ID` et trois types de trames sont disponibles :

- les messages de type **CON** pour *confirmable* indiquent qu'ils doivent être acquitté par le récepteur.
- les message de type **ACK** contiennent cet acquittement, le champ `message ID` du message à acquitter est recopié dans ce message. Ce message peut également contenir des données.
- les message de type **NON** pour *non confirmable* sont de purs datagrammes, ils ne seront pas acquittés par le récepteur, leur perte ne sera pas détectée par le protocole CoAP. Par contre, le champ `message ID` permet de détecter des messages dupliqués.

La notion d'émetteur/récepteur est dissociée des rôles de client ou de serveur définis par REST. Un client REST peut émettre des trames **CON**, **NON** ou **ACK**, de même pour un serveur.

Un message de type **RST** s'ajoute aux trois types précédents, il peut être émis par exemple quand un des nœuds a perdu son contexte suite à un redémarrage et ne sait plus traiter les réponses qu'il reçoit.

Paramètre	Valeur par défaut
ACK_TIMEOUT	2s
ACK_RANDOM_FACTOR	1.5
MAX_RETRANSMIT	4
MAX_LATENCY	100s
PROCESSING_DELAY	2s

TABLE 12.1 – Valeurs par défaut proposées par le [RFC 7252](#)

La figure 12.2 page ci-contre montre des échanges fiabilisés avec le protocole CoAP. Les messages de type CON impliquent un acquittement en retour. L'émetteur arme un temporisateur et à son expiration ré-émet le message. Si il reçoit un message d'acquittement contenant la même valeur dans le champ Message ID, le message est acquitté. Ce cas est illustré avec le Message ID valant 0x1234.

Si par contre, à l'expiration du temporiseur, l'acquittement n'est pas arrivé, le message initial, gardé en mémoire est retransmis. Le [RFC 7252](#) suggère un temporiseur initial de 2 secondes dont la valeur double à chaque retransmission, et 4 transmissions d'un même message sont possibles. Ces valeurs peuvent être changées pour s'adapter au contexte. Les durées de tempéroration incluent un aléa, pour éviter une synchronisation entre plusieurs émetteurs pouvant favoriser des collisions de trame.

La valeur du champ Message ID n'a de sens que pour un échange, si par exemple un récepteur reçoit les valeurs 0x1234 et 0x1236, il ne doit pas en déduire que le message 0x1235 s'est perdu. Plusieurs transmission peuvent également se dérouler en parallèle ; un émetteur n'a pas besoin d'attendre un acquittement pour envoyer la trame suivante.

Les messages non confirmés (type NON) utilisent également un champ message ID différent à chaque nouveau message. Il permet de détecter des duplications qui pourraient survenir dans les couches protocolaires inférieures lors du transport du message.

Pour rejeter les doublons², le récepteur doit garder une copie des Message ID émis par une source. Un simple calcul permet de définir la période de rétention des valeurs.

Un peu d'algèbre élémentaire permet de calculer cette durée. La figure 12.3 page 157 montre le calcul du pire cas. Il s'obtient quand toutes les messages CON se perdent et sont retransmis et que seul le dernier est acquitté. Comme la durée de déclenchement du temporiseur est doublée à chaque tentative, le temps passé dans cette étape est

$$(1 + 2 + 4 + \dots) \times ACK_TIMEOUT$$

ou

$$2^{MAX_RETRANSMIT} - 1 \times ACK_TIMEOUT$$

2. dus aux duplications des couches inférieures, aux pertes des messages d'acquittement forçant une réémission (cas des messages ID 0x1235 de la figure 12.2 page précédente) ou à des temporiseurs mal dimensionnés déclenchant une réémission avant la réception de l'acquittement.

Paramètre	Valeurs déduites
MAX_TRANSMIT_SPAN	45s
MAX_RTT	202s
EXCHANGE_LIFETIME	247s
NON_LIFETIME	145s

TABLE 12.2 – Valeurs déduites à partir des paramètres par défaut proposées par le [RFC 7252](#)

Pour éviter les synchronisations entre les noeuds, la valeur du temporisateur est multiplié par un facteur *ACK_RANDOM_FACTOR* compris entre 1 et 1.5. Comme on se place dans le cas le plus défavorable, on prend la valeur maximale.

On en déduit la valeur d'attente maximale avant une transmission correcte *MAX_TRANSMIT_SPAN* qui est de 45 secondes.

Une fois le message transmis, il doit arriver à destination. Le [RFC 7252](#) prend une valeurs très importante de 100s pour la latence entre l'objet et le serveur³ et 2s pour le temps de traitement. Le temps d'aller retour ou Round Trip Time (RTT) maximal est de 202s.

Pour les messages **CONFIRMÉS**, le temps maximal *EXCHANGE_LIFETIME* est donc de 245s. Pour les message **NON** confirmé, on peut supposer qu'un message sera transmis plusieurs fois pour s'assurer qu'il a été correctement reçu par le destinataire. On retrouve le même calcul sans la partie acquittement, d'où une durée de 145s.

Le standard prévoit que, par défaut, la durée d'activité d'un Message ID est d'environ 5 minutes (247 s) pour les messages confirmés, et 2,5 minutes (145 s) pour les messages non confirmés. Avoir cette notion en tête peut vous éviter des heures de débogage. Supposons qu'un client commence toujours par numérotier ses messages à 1. Le récepteur va donc garder les valeurs des messages ID pendant 5 minutes. Si vous redémarrez l'objet, il va émettre de nouveaux messages, mais avec les mêmes valeurs de Messages ID qui seront acquittés, mais pas traités ignorés par le récepteur.

12.2.3 Les Token

CoAP utilise le protocole UDP pour communiquer. Contrairement à TCP, il n'y a pas de notion d'établissement de connexion. Il est donc difficile de faire le lien entre les établissements et les réponses, surtout si elles ne sont pas immédiates. La figure suivante illustre ce phénomène. Une requête GET est envoyée par un client à un serveur.

La réponse ne peut pas être immédiate (par exemple il faut lire une valeur sur un capteur qui demande d'être active). Le message d'acquittement ne peut pas être différé sinon, le client ne voyant pas sa requête acquittée, la retransmettrait. Le serveur acquitte avec un message Ack vide (cf. figure 12.4 page ci-contre). Quand le serveur peut envoyer la ressource, il le fait à son tour dans un message de type CON qui sera à son tour acquitte. Vous pouvez remarquer que les



3. Le schéma figure 12.3 page suivante n'est pas à l'échelle.

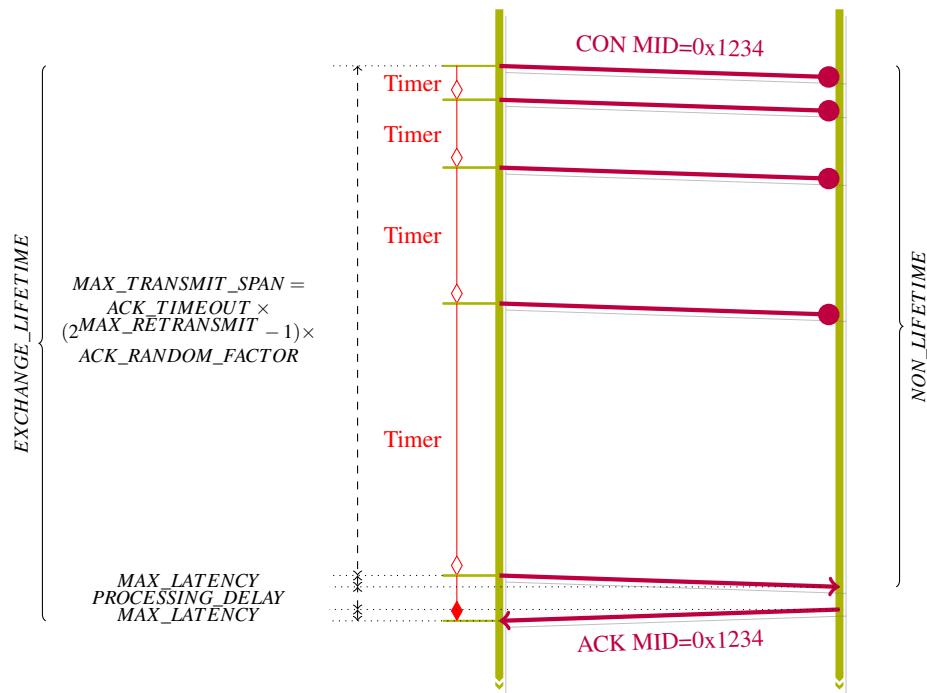


FIGURE 12.3 – Échanges fiabilisés avec CoAP

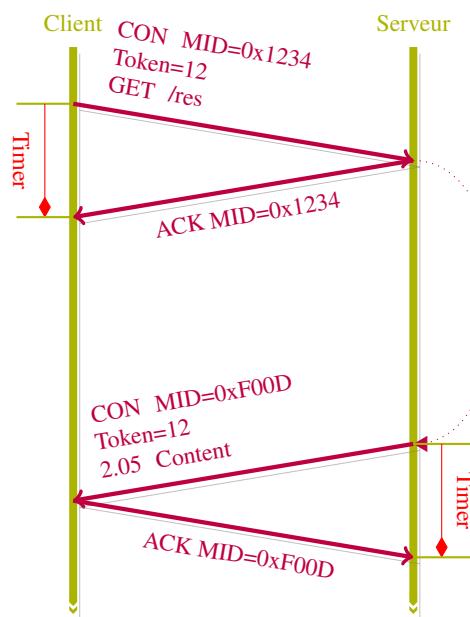


FIGURE 12.4 – Utilisation du Token

valeurs du champ Message ID sont complètement décorrélées. Pour faire le lien entre la requête et la réponse, un token fourni par le client est recopié par le serveur. C'est pour cela que l'on peut considérer une valeur de Token comme une "connexion" entre le client et le serveur.

Le Token est une séquence binaire facultative dont la taille est comprise entre 0 (pas de token) et 8 octets. La longueur est indiquée au début de l'en-tête dans le champ Token Length (TKL) et la valeur suit immédiatement l'en-tête obligatoire avant les options.

On voit bien sur cet exemple, figure 12.4 page précédente la décorrélation entre la machine protocolaire de bas niveau basée sur les Message ID et la machine protocolaire REST. Le client envoie un message CON contenant sa requête et reçoit la réponse dans un message ACK. Dans cet exemple, il y a aussi deux niveau d'acquittement au niveau des messages et avec les notifications REST.

12.2.4 Les options CoAP

Le champ Option va contenir des **options** qui vont soit servir à améliorer le protocole de transfert des données entre le client et le serveur, soit servir à coder les en-têtes des requêtes et des réponses en garantissant une certaine compatibilité avec les en-têtes HTTP.

La structure utilisée (cf. figure 12.5 page ci-contre) est dite Type Length Value (TLV) ou Type Longueur Valeur. Chaque champ contient au moins ces deux informations :

- Type indiquant la nature de l'option ;
- Longueur indiquant la taille des données en octets. Si ce dernier n'est pas nul, les données vont se trouver après.

CoAP complique un peu la chose en optant pour un codage différentiel de la valeur de l'option. Ainsi, si l'on doit envoyer une option de type 5 puis deux de type 6, le codage contiendra $\Delta T = 5$, $\Delta T = 1$, $\Delta T = 0$.

Mais comme le champ ΔT ne fait que 4 bits, on ne peut pas aller bien loin pour coder ces valeurs. Un mécanisme d'échappement est mis en place pour les différences supérieures à 13. Dans ce cas, la valeur 13 est mise dans le champ T et l'octet suivant code la différence moins 13.

Par exemple, si l'on doit coder deux options de valeurs 5 et 20, la différence est de 15. La première option est codée normalement avec le ΔT à 5. Pour la seconde option, le ΔT est mis à 13 et l'octet suivant prendra la valeur 2.

Notez que la valeur 14 mise dans le champ ΔT indique que la différence nécessite deux octets pour être codée. Les principales options utilisés dans CoAP se retrouvent listés dans le tableau 12.3 page 161. Celles apparaissant sur fond bleu, seront traitées plus en détail dans cet ouvrage.

Pour la longueur on retrouve le même principe : les longueurs inférieures à 13 sont codées directement ; si elles sont supérieures ou égale à 13, la valeur moins 13 est codée dans un octet supplémentaire. Une valeur de 14 indique que deux octets sont utilisés pour coder la longueur moins 269.

La figure 12.5 page ci-contre illustre le codage d'une option dans l'en-tête d'un message CoAP.

Il se peut qu'il y ait des données après les options. Dans ce cas, un séparateur avec la valeur 0xFF est inséré. Il ne peut pas être confondu avec le codage d'une option puisque les champs ΔT et Longueur n'évoluent qu'entre 0 et 14.



FIGURE 12.5 – Format des options

S'il n'y a pas de données à transmettre (par exemple dans le cas d'une requête GET), le message CoAP se termine après les options.

12.2.5 Options CoAP

Le premier bit servant à coder le type décrit, quand il est positionné à 1, si ce type doit être connu du récepteur (critique). Dans ce cas, si un destinataire reçoit une option de ce type et qu'il ne la connaît pas, il doit produire un message d'erreur. Dans le cas contraire cette option est ignorée du récepteur qui poursuit le traitement des options suivantes. Ainsi les options paires sont facultatives et les impaires critiques.

12.2.6 Représentation des URI

On comprend mieux la signification de certaines options données dans le tableau précédent quand la syntaxe d'un URI est connue. (voir [RFC 2396](#)) . On a déjà vu que la syntaxe générale est :

```
<schema>:<schema-specific-part>
```

où *schema* va définir le schéma de notation. De manière générale, le schéma va indiquer comment est structuré la suite de l'URI. Quand l'URI est aussi un localisateur (donc un URL), le schéma fait référence au protocole qui pourra être utilisé pour retrouver la ressource comme http ou https voire coap. Après le schéma, on trouve deux zones, l'autorité qui va indiquer qui est responsable de nommer les ressources. Dans le cas d'un URL, il peut s'écrire de la manière suivante :

```
<schema>://userinfo@host@:port@/path?query
```

où les champs en italique *userinfo@* et *:port* sont facultatifs. Ils contiennent respectivement le nom de l'utilisateur et le numéro de port sur lequel tourne le service.

path va être composé d'une série de segments séparés par des caractères/ qui identifient la ressource sur le serveur. L'URI peut se terminer par des questions, c'est-à-dire une chaîne de caractères qui sera interprétée par le serveur précédemment désigné. Une question, c'est-à-dire des paramètres fournis pour construire la ressource, peut contenir plusieurs parties séparées par le caractère .

Ceci peut être vérifié par le programme de désassemblage. L'URI :

```
coap://192.168.1.52/capteur1/temperature?max_value&date=20131206
```

utilise le schéma de nommage de coap. Le serveur est 192.168.1.52, le chemin (*path*) est composé de deux segments, suivi par deux questions. Le serveur reçoit la requête suivante :

```
Received packet of size 50
40 01 BE BF|B8 63 61 70 74 - 65 75 72 31|0B 74 65 @....capt eur1.te
6D 70 65 72 61 74 75 72 65 -|49 6D 61 78 5F 76 61 mperature Imax_va
6C 75 65|0D 00 64 61 74 65 - 3D 32 30 31 33 31 32 lue..date =201312
30 36 06
ver:1 Type = 0 (CON) Token Length = 0 code 1 (GET) Msg id = BEBF
Option = 11 (+11) length = 8
Uri-Path capteur1
```



Valeur	Nom	Type	Nature	répété	Commentaire
0	Reservé				
1	If-Match	opaque	critique	oui	Utilisé pour indiquer à un serveur de ne pas effectuer la requête que sous certaines conditions.
3	Uri-Host	string	critique		Contient le nom du serveur d'une URI (nom, adresse IPv4 ou IPv6). Généralement, il n'est pas nécessaire de le préciser puisque les messages CoAP sont envoyés à cette adresse.
4	ETag	opaque	facultative	oui	Utilisé pour gérer la mise en cache des ressources
5	If-None-Match	vide	critique		Utilisé pour indiquer à un serveur de ne pas effectuer la requête que sous certaines conditions.
6	Observe	entier	facultative		Permet à un serveur d'envoyer une requête aux changements d'état d'une ressource. Dans la réponse la valeur doit toujours augmenter.
7	Uri-Port	entier	critique		Contient le numéro du port UDP sur lequel CoAP est lancé. Généralement ce champ n'est pas nécessaire vu que le serveur CoAP attend déjà des messages sur ce port.
8	Location-Path	string	facultative	oui	Utilisé en réponse à une requête POST pour indiquer un segment du chemin de la ressource.
11	Uri-Path	string	critique	oui	Contient un ou plusieurs segments de l'URI
12	Content-Format	entier	facultative		Définit le format dans lequel sont codées des données
14	Max-Age	entier	facultative		Durée pendant laquelle la ressource peut être mise en cache.
15	Uri-Query	string	critique	oui	Contient les segments d'interrogation que l'on retrouve dans les URIs. Contains the query segments found in URIs.
17	Accept	entier	critique		Indique les formats que le client peut accepter.
20	Location-Query	string	facultative	oui	Utilisé en réponse à une requête POST pour indiquer le chemin de la ressource.
35	Proxy-Uri	string	critique		Contient une URI qui doit être prise en compte par le proxy.
39	Proxy-Scheme	string	critique		Indique le schéma d'encodage.
60	Size1	entier	facultative		Indique la taille de la ressource.
258	No-Response	entier	facultative		Limite les notifications REST

TABLE 12.3 – Certaines options du protocole CoAP

```
Option = 11 (+0) length = 11
Uri-Path temperature
```

```
Option = 15 (+4) length = 9
Uri-Query max_value
Option = 15 (+0) length = 13
Uri-Query date=20131206
```

Le listing précédent montre ce que reçoit le serveur. L'URI n'est pas complète, car la partie qui a servi à le localiser n'est pas indispensable. Seules les parties "chemin" et "question" sont indiquées. Le schéma coap : n'est pas précisé ; de même que Uri-Host et Uri-Port car le serveur connaît son adresse IP et le numéro de port sur lequel s'exécute le serveur CoAP.

Ils pourraient être utiles en cas de virtualisation du serveur, c'est-à-dire si plusieurs instances de CoAP tournaient, soit à des noms différents, soit sur des numéros de port différents. Si cette possibilité existe, pour l'instant, la faible capacité des ressources ne pousse pas vers une virtualisation.

Si on reprend la partie optionnelle, la première option qui commence par l'octet 0xB8. 0xb (=11) indique qu'il s'agit d'une option **Uri-path** (comme c'est la première option, le delta se confond avec la valeur de l'option) et de longueur 8 octets qui correspondent à la valeur capteur1. La seconde option débute par 0x0B. Le delta est nul. On reste sur une option Uri-path de longueur de 11 octets. La troisième option s'ouvre avec l'octet 0x49. L'incrément étant de 4, le numéro de l'option passe à 15 soit **Uri-query** avec une valeur sur 9 octets. L'option suivante démarre par 0x0D. On reste sur une option Uri-query mais la longueur 0xD informe que l'octet suivant contient la longueur. La valeur vaut étrangement 0x00 car elle est diminuée de 13 pour respecter le codage défini par CoAP. la longueur est donc de 13 octets.

Questions sur les URI

Soit le message CoAP suivant :

```
40020001b474656d700573656e7331436d6178ff32332e30
```

Question 12.2.1: Code ?

Que représente ce message ?

- Une requête GET
- Une requête POST
- Une requête PUT
- Une requête DELETE
- Une notification positive

Question 12.2.2: Token or not Token ?

Quelle est la valeur du champ token ?

- Vide
- 0xb4
- 0xb474
- 0xb47465
- 0xb474656d

Question 12.2.3: chemin d'URI

Quels éléments de l'URI contient ce message ?

- Aucun
- /temp
- /temp/sens1 et /max
- /temp/sens1/max
- /temp/sens1?max

12.2.7 Représentation des données

Pour que le client puisse interpréter les données, il faut qu'il puisse comprendre comment elles sont représentées. Cela peut dépendre de la police de caractères. Ainsi, une lettre accentuée ne sera pas représentée de la même manière suivant le type de code. Il en va de même pour la représentation. Le plus simple consiste à envoyer en ASCII la valeur demandée, par exemple la chaîne de caractères 18 indique 18 degrés. Il faut donc indiquer le type de codage/sérialisation utilisé pour décrire le contenu de la ressource. Là où HTTP utiliserait un nom, c'est-à-dire une chaîne de caractères, CoAP va utiliser une valeur numérique.

Le tableau précédent donne un extrait des valeurs utilisées pour représenter les formats⁴.

Deux options CoAP utilisent ces codes :

- **Content-format** (12) indiquant comment la ressource est codée ;
- **Accept** (17) indiquant dans une requête le format dans lequel la réponse doit être codée.

Il existe beaucoup de valeurs pour le content-format, elles permettent de spécifier de manière très économique le type de ressource et par conséquent le traitement à effectuer.

Chaque protocole utilisant CoAP aura tendance à définir de nouveaux codes. Le tableau 12.4 illustre ce phénomène pour **SenML**. La valeur va servir à la fois pour indiquer la structure de données et le format de codage.

Valeur	Type
0	text/plain ; charset=utf-8
40	application/link-format
41	application/xml
42	application/octet-stream
47	application/exi
50	application/json
60	application/cbor
110	application/senml+json
112	application/senml+cbor
11542	application/vnd.oma.lwm2m+tlv
11543	application/vnd.oma.lwm2m+json

TABLE 12.4 – Type des données

4. La liste complète peut être trouvée sur le site de l'IANA <https://www.iana.org/assignments/core-parameters/core-parameters.xhtml#content-formats>.

Question 12.2.4: ASCII

Vous avez la ressource suivante :

`temperature = 20C`

Quelle valeur l'option CoAP Content-type utiliser pour une réponse en CoAP ?

- text/plain
- 0
- 50

Question 12.2.5: SenML

Vous voulez recevoir une ressource dans le format SenML; CBOR. Quelle valeur doit transporter l'option Accept dans la requête ?

Question 12.2.6: Erreur

Quel code d'erreur retourne le serveur s'il ne peut pas envoyer une réponse dans ce format ?

Aidez vous du [RFC 7252](#).

- 4.04 (Not Found)
- 4.02 (Bad Option)
- 4.06 (Not Acceptable)
- 5.01 (Not Implemented))

12.3 Observe

Avec l'architecture REST, le serveur répond toujours aux requêtes d'un client. Si l'on veut suivre l'évolution d'une ressource, le client doit demander périodiquement la valeur; le serveur ne gardant pas d'état sur les requêtes passées. Cela n'est pas toujours compatible avec les contraintes énergétiques des capteurs. Supposons que l'on ait une alarme d'incendie qui doit informer quand le taux de fumée atteint un certain seuil. Il existe deux possibilités :

- l'alarme est un client REST et envoie un POST vers un serveur quand l'alerte est déclenchée. Pour que cela fonctionne, il faut que l'alarme ait été préalablement configurée avec l'adresse du serveur vers où envoyer ses requêtes POST;
 - l'alarme est un serveur REST qui possède une ressource donnant le taux de fumée. Elle n'a pas besoin d'être configurée. Les clients l'interrogent et elle répond à leur adresse. En revanche, si l'on veut déterminer quand un seuil est atteint, il faut continuellement interroger la ressource à une fréquence élevée si l'on ne veut pas rater une information.
- the alarm is a REST server that has a resource giving the smoke rate. It does not need to be configured. Clients query it and it responds to their address. However, if you want to determine when a threshold is reached, you have to continuously query the resource at a high frequency if you don't want to miss any information.*

L'option **Observe**, définie dans le [RFC 7641](#), permet à un serveur d'envoyer périodiquement la valeur d'une ressource vers le client qui en a fait la demande. La période d'émission (ou les règles d'émission comme envoyer quand un seuil est atteint) est définie par le comportement du serveur.

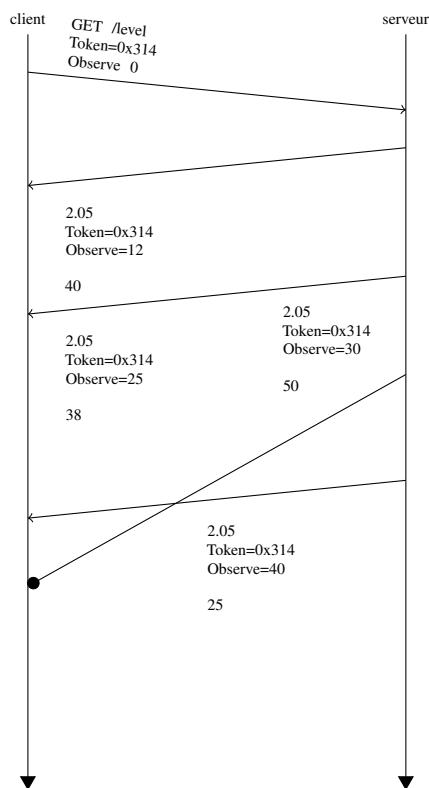


FIGURE 12.6 – Envoi périodique grâce à l'option Observe

La figure 12.6 page précédente illustre ce comportement. Le client envoie une requête GET en positionnant l'option Observe avec la valeur 0, mais dans valeur. Si le client accepte cette option, il va répondre en la positionnant dans ses réponses. Elle doit dans ce cas comporter une valeur qui ne pourra que croître de réponse en réponse. Cela est utile pour permettre au client de détecter un déséquancement des réponses. Ainsi dans l'exemple, l'Observe estampillé 30 arrive après celui estampillé 40 et sera rejeté par le client.

On notera également que le champ **Token** doit être présent pour faire le lien entre la requête et les réponses.

Il faut pouvoir aussi arrêter un Observe. Il existe plusieurs cas de figure :

- le client veut stopper un Observe en cours. Il refait la même requête mais en mettant la valeur 1 dans l'option Observe.
- le client redémarre, il va perdre peut perdre son contexte concernant l'Observe, mais continuer à recevoir périodiquement des requêtes provenant du serveur. Le client ne reconnaissant pas le Token, émet un message **ReSeT**. Le serveur annule l'émission périodique vers le client.
- le client est inaccessible, il ne va pas pouvoir annuler la transmission. En règle générale, les réponses avec l'option Observe sont transportées dans des messages **NON** confirmables. Le serveur peut de temps en temps envoyer la réponse dans un message **CON**firmeable. S'il ne reçoit pas d'acquittement du client, il en déduit qu'il a disparu et arrête d'envoyer des réponses périodiques.

13. Experimentons CoAP

Les programmes se trouvent dans le répertoire `pycom` pour l'Objet et `plido-tp4` pour le serveur.

Les programmes pour l'Objet pourront tourner indifféremment dans une fenêtre de votre ordinateur ou sur votre Pycom. Dans la suite, nous supposerons qu'il s'agit d'une deuxième fenêtre de terminal sur votre ordinateur mais libre à vous de le lancer sur votre Pycom en Wi-Fi (voire en LoRaWAN) pour plus de réalisme. L'utilisation du réseau Sigfox est plus problématique vu que les requêtes provenant de votre Pycom sont limitées à 12 caractères et que les réponses ne sont qu'au nombre de 4 par jour et limitées à 8 caractères.

Pour nous concentrer sur le fonctionnement de CoAP, nous allons tout d'abord expérimenter en local sur notre ordinateur. Nous verrons par la suite comment utiliser le programme `generic_relay.py` pour bénéficier des réseaux LPWAN..

Attention, le serveur ne fonctionne pas directement sous Windows. Vous devez impérativement le faire tourner dans la machine virtuelle.

13.1 Mise en œuvre du client/serveur

Nous allons mettre en œuvre le protocole CoAP entre deux processus sur votre machine puis, si vous le voulez, vous pourrez le tester sur un LoPy. Côté Objet, nous allons utiliser une mise en œuvre simple mais compacte du protocole pour comprendre son fonctionnement. À l'autre extrémité, nous utiliserons la mise en œuvre `aiocoap` qui est très complète mais beaucoup plus complexe et demandant plus de ressources.

Youtube



13.1.1 aiocoap

Comme son nom l'indique, `aiocoap` met en œuvre CoAP, avec les modules asynchrones Input/Output, permettant un fort degré de parallélisme. Le programme suivant (`coap_basic_server1.py`) donne un exemple d'un serveur simple gérant une seule ressource time :

Listing 13.1 – coap_basic_server1.py

```

1 #!/usr/bin/env python3

3 # This file is part of the Python aiocoap library project.
#
5 # Copyright (c) 2012-2014 Maciej Wasilak <http://sixpinetrees.blogspot.com/>,
#                   2013-2014 Christian Amsüss <c.amsuess@energyharvesting.at>
7 #
# aiocoap is free software, this file is published under the MIT license as
9 # described in the accompanying LICENSE file.

11 """This is a usage example of aiocoap that demonstrates how to implement a
13 simple server. See the "Usage Examples" section in the aiocoap documentation
15 for some more information."""
16

17 import datetime
18 import logging
19 import socket

20 import asyncio
21
22 import aiocoap.resource as resource
23 import aiocoap

24
25 class TimeResource(resource.Resource):
26
27     async def render_get(self, request):
28         await asyncio.sleep(5)

29         payload = datetime.datetime.now().\
30             strftime("%Y-%m-%d %H:%M").encode('ascii')
31
32         return aiocoap.Message(payload=payload)

33 # logging setup

34
35 logging.basicConfig(level=logging.INFO)
36 logging.getLogger("coap-server").setLevel(logging.DEBUG)
37

38 def main():
39     # Resource tree creation
40     with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as s:
41         s.connect(("8.8.8.8", 80)) # connect outside to get local IP address
42         ip_addr = s.getsockname()[0]

43     port = 5683
44     print ("server running on", ip_addr, "at", port)

45     root = resource.Site()

46     root.add_resource(['time'], TimeResource())

47     asyncio.Task(
48         aiocoap.Context.create_server_context(root,
49                                             bind=(ip_addr, port)))

50
51     asyncio.get_event_loop().run_forever()

```

```
57 if __name__ == "__main__":
    main()
```

- Lignes 21 et 22, l'utilisation de *aicoap* se traduit par l'importation des modules qui se trouvent dans le répertoire *aoicoap*.
- Dans la fonction *main* le programme :
 - cherche son adresse IP fixe (lignes 40 à 42) ;
 - fixe à la valeur par défaut le numéro de port à la valeur affectée au serveur CoAP (ligne 44)¹ ;
 - ligne 47, La variable *root* contient l'arbre des ressources. À la ligne suivante, la ressource *time* est associée à une classe *TimeResource*.
 - La méthode :


```
asyncio.Task(aiocoap.Context.create_server_context(root, bind=(ip_addr, port)))
```

 permet de lier cet arbre de ressource à l'adresse IP et au numéro de port précédemment défini.
 - Ligne 55, le serveur est ensuite lancé dans une boucle sans fin.

Il est plus intéressant de voir le traitement effectué lorsque la ressource est appelée par le serveur. La classe *TimeResource* dérivant de la classe générique *aiocoap Resource* est utilisée (ligne 24) :

```
class TimeResource(resource.Resource):
```

Pour chaque méthode CoAP, une méthode peut être définie dans cette classe. Dans l'exemple, la méthode *render_get* permet de traiter les requêtes GET.

Pour simuler un temps de traitement, le programme commence par attendre 5 secondes (ligne 27) puis construit la chaîne de caractères contenant la date qu'il va retourner dans un objet *aoicoap Message*.

Ainsi, si tout se passe correctement, la réponse à une requête (Code = 0x01) sera 2.05 (Content). La figure 13.1 page suivante résume cet échange que nous avions vu théoriquement dans le chapitre 12.2.3 page 156 consacré aux tokens.

13.1.2 côté Objet

Du côté du client, nous allons utiliser une mise en œuvre plus compacte qui nous permettra d'expérimenter le fonctionnement de CoAP en modifiant les valeurs des champs protocolaires.

Listing 13.2 – coap_empty_msg.py

```
1 import CoAP
2 import socket
3
4 SERVER = "192.168.1.XX" # change to your server's IP address
5 PORT   = 5683
6
7 s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
```

1. L'utilisation d'une adresse IP et non du joker 0.0.0.0 permet de faire tourner le serveur dans un environnement Windows et MAC

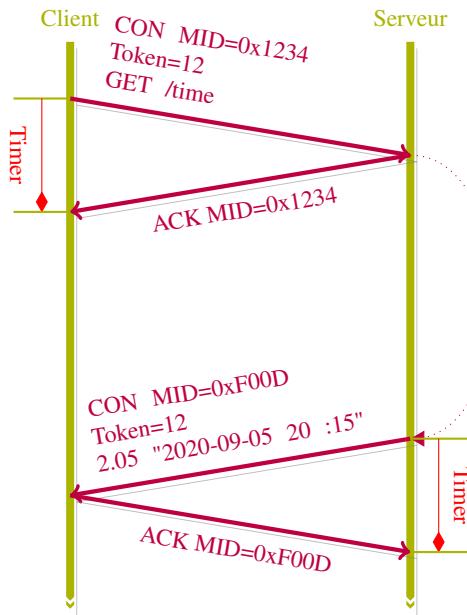


FIGURE 13.1 – Récupération de la date

```

9 coap = CoAP.Message()
10 coap.new_header()
11 coap.dump(hexa=True)

13 s.settimeout(10)
14 s.sendto(coap.to_byte(), (SERVER, 5683))

15
16 resp,addr = s.recvfrom(2000)
17 answer = CoAP.Message(resp)
18 answer.dump(hexa=True)

```

Le programme `coap_empty_msg.py` est beaucoup plus simple. Dans un premier temps, vous devez remplacer l'adresse IP par celle fournie par votre serveur CoAP. Le programme crée une socket UDP au travers de laquelle l'échange avec le serveur sera effectué. La première action consiste à créer un message CoAPCoAP (ligne 9) et à la ligne suivante créer un en-tête obligatoire avec les paramètres par défaut avec la fonction CoAP.

Le programme affiche le message avec la fonction `CoAPdump`(ligne 11) et l'envoie sur la socket. La ligne 13 permet de limiter l'attente de la réponse à 10 secondes. Cette réponse est attendue ligne 16, transformée en message CoAP ligne 17 et affichée.

Lancez une capture Wireshark pour voir le trafic passant sur le port de CoAP (`udp.port==5683` dans la fenêtre de filtrage).

Lancez maintenant le programme client.

```

b      40000001
CON 0 x0001 EMPTY
b      70000001

```

```
RST 0 x0001 EMPTY
```

Les messages suivants ont circule sur le réseau.

```
18:38:30.381449 IP 192.168.1.26.50883 > 192.168.1.26.5683: UDP, length 4
 0x0000: 4500 0020 221f 0000 4011 0000 c0a8 011a E..."@.....
 0x0010: c0a8 011a c6c3 1633 000c 83a2 4000 0001 .....3....@...
18:38:30.382107 IP 192.168.1.26.5683 > 192.168.1.26.50883: UDP, length 4
 0x0000: 4500 0020 efbb 0000 4011 0000 c0a8 011a E.....@.....
 0x0010: c0a8 011a 1633 c6c3 000c 83a2 7000 0001 .....3.....p...
```

On retrouve dans le contenu des messages UDP, le message CoAP donné par l'application. Si l'on prend le deuxième message, il commence par 0x70; ce qui correspond en binaire à 0b01_11_0000, soit version = 1, type = 3 et longueur du token = 0. L'octet suivant donne le code 0 (**Empty**) et les deux derniers octets contiennent le message ID.

Le serveur ne sachant pas quoi faire de la requête, la rejette en envoyant un message ReSeT pour essayer d'arrêter le code sur le client qui envoie ce genre de requête.

13.2 GET /time

Nous laissons tourner le serveur CoAP et nous allons construire la requête CoAP du client pour qu'il demande la ressource `/time`².

Listing 13.3 – coap_get_time.py

```
import CoAP
import socket

s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

coap = CoAP.Message()
coap.new_header()
coap.new_header(code=CoAP.GET, mid=0xF001)
coap.add_option(CoAP.Uri_path, "time")
coap.dump()

s.settimeout(10)
s.sendto(coap.to_byte(), ("192.168.1.77", 5683))

resp, addr = s.recvfrom(2000)
answer = CoAP.Message(resp)
answer.dump()

s.settimeout(10)
resp, addr = s.recvfrom(2000)
answer = CoAP.Message(resp)
answer.dump()
```

la méthode `new_header` précise le code, ici GET et on ajoute l'élément d'URI time en option (ligne 10) grâce à la méthode `add_option`. Côté client, on obtient le résultat suivant :

```
> python3 coap_get_time1.py
False
b'40010001b474696d65'
CON 0x0001 GET
```

2. N'oubliez pas de mettre la bonne adresse IP du SERVER dans ce programme et dans les suivants

1208 169459.0845s 192.168.1.79	192.168.1.79	CoAP	51 CON, MID:1, GET, /time
1209 169459.1876s 192.168.1.79	192.168.1.79	CoAP	46 ACK, MID:1, Empty Message
1210 169464.0933s 192.168.1.79	192.168.1.79	CoAP	63 CON, MID:19224, 2.05 Content
1211 169464.0933s 192.168.1.79	192.168.1.79	ICMP	91 Destination unreachable (Port unreachable)

FIGURE 13.2 – Échange incomplet

```
> Uri-path : b'time',
b'60000001',
ACK 0x0001 EMPTY
```

La requête CoAP commence par le mot 40010001, indiquant un message **CON**firmeable, sans Token, un code GET et un MID de 0x0001, suivi de l'option **Uri-Path**.

La réponse est un **ACK** avec la même valeur de *Message ID* et le code est vide (0.00).

On n'obtient pas la réponse au GET, juste un acquittement. Pourtant, les log du serveur et l'analyse du réseau montrent bien que le serveur à répondu.

```
DEBUG:coap-server:Incoming message <aiocoap.Message at 0x7f3d997ace80: Type.CON GET (MID 1, empty token) remote
<UDP6EndpointAddress 192.168.1.79:52495 (locally 192.168.1.79%lo)>, 1 option(s)
DEBUG:coap-server:New unique message received
DEBUG:coap-server:Sending empty ACK: Response took too long to prepare
DEBUG:coap-server:Sending message <aiocoap.Message at 0x7f3d99742748: Type.ACK EMPTY (MID 1, empty token) remote
<UDP6EndpointAddress 192.168.1.79:52495 (locally 192.168.1.79%lo)>>
```

Les trames qui ont circulé sur le réseau sont représentées figure 13.2. Comme nous avons ajouté un délai de 5 secondes avant de répondre sur le serveur CoAP dans la méthode `render_get`, le serveur acquitte la requête et cherche 5 secondes plus tard à envoyer une nouvelle requête confirmée. Mais le client, ayant fermé sa socket, ne peut plus la recevoir et retourne une erreur ICMP.

La solution est d'attendre un second message et de le décoder comme le montre le listing suivant qui donne le code du programme `coap_get_time2.py` dans lequel vous n'aurez pas oublié de changer l'adresse IP du serveur.

Listing 13.4 – `coap_get_time2.py`

```
import CoAP
import socket

SERVER = "192.168.1.XX" # change to your server's IP address
PORT = 5683

s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

coap = CoAP.Message()
coap.new_header(code=CoAP.GET)
coap.add_option(CoAP.Uri_path, "time")
coap.dump()

s.sendto(coap.to_byte(), (SERVER, PORT))

s.settimeout(10)
resp, addr = s.recvfrom(2000)
answer = CoAP.Message(resp)
answer.dump()

s.settimeout(10)
```

1299	170334.4503:	192.168.1.79	192.168.1.79	CoAP	51 CON, MID:1, GET, /time
1300	170334.5545:	192.168.1.79	192.168.1.79	CoAP	46 ACK, MID:1, Empty Message
1307	170339.4599:	192.168.1.79	192.168.1.79	CoAP	63 CON, MID:9992, 2.05 Content
1308	170342.2983:	192.168.1.79	192.168.1.79	CoAP	63 CON, MID:9992, 2.05 Content
1309	170342.2983:	192.168.1.79	192.168.1.79	ICMP	91 Destination unreachable (Port unreachable)

FIGURE 13.3 – Échange encore incomplet

```

22 resp, addr = s.recvfrom(2000)
23 answer = CoAP.Message(resp)
24 answer.dump()

```

Nous pouvons laisser éclater notre joie car nous avons la réponse :

```

> python3 coap_get_time2.py
False
CON 0x0001 GET
> Uri-path : b'time'
ACK 0x0001 EMPTY
CON 0x2708 2.05
---CONTENT
hex: b'323032312d30332d33302031343a3537',
txt: b'2021-03-30 14:57'

```

Mais notre joie est de courte durée car si on regarde plus attentivement la figure 13.3 le trafic sur le réseau, on voit que la réponse a été émise deux fois et que l'on retrouve ensuite une erreur ICMP. Cela est dû au fait que l'on n'acquitte pas le message venant du serveur. Le croyant perdu, il le retransmet et tombe sur une socket inexistante.

Pour bien faire, nous devons fournir un acquittement au serveur avec ce code de client avec le programme `coap_get_time3.py`.

Listing 13.5 – `coap_get_time3.py`

```

1 import CoAP
2 import socket
3
4 SERVER = "192.168.1.XX" # change to your server's IP address
5 PORT = 5683
6
7 s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
8
9 coap = CoAP.Message()
10 coap.new_header(code=CoAP.GET)
11 coap.add_option(CoAP.Uri_path, "time")
12 coap.dump()
13
14 s.sendto(coap.to_byte(), (SERVER, PORT))
15
16 s.settimeout(10)
17 resp,addr = s.recvfrom(2000)
18 answer = CoAP.Message(resp)
19 answer.dump()
20
21 s.settimeout(10)
22 resp,addr = s.recvfrom(2000)

```

1409 171332.3768f 192.168.1.79	192.168.1.79	CoAP	51 CON, MID:1, GET, /time
1410 171332.4796f 192.168.1.79	192.168.1.79	CoAP	46 ACK, MID:1, Empty Message
1411 171337.3847f 192.168.1.79	192.168.1.79	CoAP	63 CON, MID:9993, 2.05 Content
1412 171337.3849f 192.168.1.79	192.168.1.79	CoAP	46 ACK, MID:9993, Empty Message

FIGURE 13.4 – Échange dans les règles de l’art

```

23 answer = CoAP.Message(resp)
24 answer.dump()
25
26 mid = answer.get_mid()
27 ack = CoAP.Message()
28 ack.new_header(mid=mid, type=CoAP.ACK)
29 ack.dump()
30 s.sendto(ack.to_byte(), (SERVER, PORT))

```

Ici, l’exécution est parfaite :

```

> python3 coap_get_time3.py
False
CON 0x0001 GET
> Uri-path : b'time'
ACK 0x0001 EMPTY
CON 0x2709 2.05
---CONTENT
hex: b'323032312d30332d33302031353a3133',
txt: b'2021-03-30 15:13'
ACK 0x2709 EMPTY

```

si vous n’avez pas oublié de changer l’adresse IP du serveur, et l’utilisation du réseau est optimale (cf figure 13.4).

Enfin, on peut ajouter un token pour lier les deux transactions. Ici, il n’y a pas d’ambiguïté car nous ne demandons qu’une seule ressource. Mais si nous demandions plusieurs ressources simultanément, il faudrait pouvoir associer la réponse à la requête. Le programme `coap_get_time4.py` ajoute lors de la création de l’en-tête un champ token.

Listing 13.6 – `coap_get_time4.py`

```

coap = CoAP.Message()
10 coap.new_header(code=CoAP.GET, token=0x12345)
11 coap.add_option(CoAP.Uri_path, "time")
12 coap.dump()

```

L’échange est le même mais le token est répété dans la réponse.

```

> python3 coap_get_time4.py
False
CON 0x0001 GET T=012345
> Uri-path : b'time'
ACK 0x0001 EMPTY
CON 0x270A 2.05 T=012345
---CONTENT
hex: b'323032312d30332d33302031353a3232',
txt: b'2021-03-30 15:22'
ACK 0x270A EMPTY

```

Question 13.2.1: NON

Que se passe-t-il si vous utilisez une requête non confirmable pour demander la ressource /time (mettre l'argument type=CoAP.NON dans la construction de l'en-tête obligatoire).

- Ça plante le serveur.
- Le serveur répond par une requête Confirmable.
- Le serveur retourne un RST car nous n'avons pas programmé ce cas.
- Le serveur répond par une requête Non Confirmable.
- On passe en heure d'hiver.

Question 13.2.2: Réponse immédiate et CON

Modifiez le programme du serveur pour supprimer le délai de 5 secondes avant une réponse.

Que se passe-t-il quand le client envoie une requête CONfirmable ?

- Le serveur retourne la réponse dans l'acquittement.
- Le serveur retourne une requête NON confirmable.
- Le serveur attend quand même quelques secondes pour ne pas saturer le réseau.
- Le serveur enlève le token de la réponse.

Question 13.2.3: Réponse immédiate et NON

Modifiez le programme du serveur pour supprimer le délai de 5 secondes avant une réponse.

Que se passe-t-il quand le client envoie une requête NON confirmable ?

- Le serveur retourne la réponse dans l'acquittement.
- Le serveur retourne une requête NON confirmable.
- Le serveur attend quand même quelques secondes pour ne pas saturer le réseau.
- Le serveur enlève le token de la réponse.

13.3 POST

13.3.1 Ressource codée en ASCII

Nous allons voir le traitement d'une requête POST avec *aiocoap*. Nous aurions pu ajouter ce traitement à la classe TimeResource vue précédemment, mais nous avons préféré ajouter une nouvelle ressource pour le traitement des températures.

Les lignes suivantes ont été ajoutées pour traiter le POST. Notez le nom de la méthode render_post.

Listing 13.7 – coap_basic_server2.py

```

34 class TemperatureResource(resource.Resource):
35     async def render_post(self, request):
36         print ("-"*20)
37         print ("payload:", binascii.hexlify(request.payload))
38         resp = aiocoap.Message(code=aiocoap.CHANGED)
39         return resp

```

La réponse à la requête est le code 2.04 (aiocoap.CHANGED) qui indique que la ressource a été modifiée.

On lui associe une nouvelle classe dans l'arbre des ressources :

```
root.add_resource(['temp'], TemperatureResource())
```

Le tout forme le programme `coap_basic_server2.py`.

Coté client, le principe est le même que pour le GET. Dans le programme `coap_post_temp1.py`, la méthode `add_payload` permet d'ajouter du contenu à la requête POST. Le programme émet sa requête et affiche le résultat. Nous allons également simplifier la gestion des réponses en utilisant la fonction `send_ack` du module CoAP. Cette fonction prend en argument une socket, un tuple de destination et le message CoAP à envoyer. Elle va le retransmettre au maximum 4 fois tant qu'elle n'a pas reçu l'acquittement correspondant.

Listing 13.8 – `coap_post_temp1.py`

```

1 import CoAP
2 import socket
3 import time
4
5 SERVER = "192.168.1.XX" # change to your server's IP address
6 PORT = 5683
7 destination = (SERVER, PORT)
8
9 s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
10
11 coap = CoAP.Message()
12 coap.new_header(code=CoAP.POST)
13 coap.add_option(CoAP.Uri_path, "temp")
14 coap.add_payload("23.5")
15 coap.dump()
16
17 answer = CoAP.send_ack(s, destination, coap)
18 answer.dump()
```

La figure 12.3 page 157 a montré un échange avec perte. Elles peuvent être simulées en arrêtant le programme serveur sur votre ordinateur.

Lancez le serveur `coap_basic_server2.py` et tapez ctrl-Z pour stopper son exécution, tout en laissant ouverte la socket.

```
> python3 ./coap_basic_server2.py
server running on 192.168.1.79 at port 5683 ^Z
Job 2,    python3 ./coap_basic_server2.py has stopped
>
```

Lancez le client `coap_port_temp1.py` (en n'oubliant pas de changer l'adresse IP du serveur).

```
> python3.9 coap_post_temp1.py
False
CON 0x0001 POST
> Uri-path : b'temp'
---CONTENT
hex: b'32332e35'
txt: b'23.5'
timeout 2 1
timeout 4 2
```

18	93.56226723:	192.168.1.79	192.168.1.79	CoAP	56 CON, MID:1, POST, /temp
19	105.5660928:	192.168.1.79	192.168.1.79	CoAP	56 CON, MID:1, POST, /temp
20	119.5771454:	192.168.1.79	192.168.1.79	CoAP	56 CON, MID:1, POST, /temp
27	122.0071288:	192.168.1.79	192.168.1.79	CoAP	46 ACK, MID:1, 2.04 Changed
28	122.0169795:	192.168.1.79	192.168.1.79	CoAP	46 ACK, MID:1, 2.04 Changed
29	122.0170019:	192.168.1.79	192.168.1.79	ICMP	74 Destination unreachable (Port unreachable)
30	122.0272629:	192.168.1.79	192.168.1.79	CoAP	46 ACK, MID:1, 2.04 Changed
31	122.0272841:	192.168.1.79	192.168.1.79	ICMP	74 Destination unreachable (Port unreachable)

FIGURE 13.5 – Trafic lié au POST

On voit que le client ne reçoit pas l’acquittement, déclenche un temporisateur et retransmet le message. La durée du temporisateur double à chaque tentative³.

Réactivez le serveur coap en tapant fg (foreground)

```
> fg
DEBUG:coap-server:Incoming message <aiocoap.Message at 0x7f05d621fcc0: Type.CON POST (MID 1, empty token) remote
<UDP6EndpointAddress 192.168.1.79:37286 (locally 192.168.1.79%lo)>, 1 option(s), 4 byte(s) payload>
DEBUG:coap-server>New unique message received
-----
payload: b'32332e35'
DEBUG:coap-server:Sending message <aiocoap.Message at 0x7f05d622df98: Type.ACK 2.04 Changed (MID 1, empty token)
remote <UDP6EndpointAddress 192.168.1.79:37286 (locally 192.168.1.79%lo)>>
DEBUG:coap-server:Incoming message <aiocoap.Message at 0x7f05d621fcc0: Type.CON POST (MID 1, empty token) remote
<UDP6EndpointAddress 192.168.1.79:37286 (locally 192.168.1.79%lo)>, 1 option(s), 4 byte(s) payload>
INFO:coap-server:Duplicate CON received, sending old response again
DEBUG:coap-server:Sending message <aiocoap.Message at 0x7f05d622df98: Type.ACK 2.04 Changed (MID 1, empty token)
remote <UDP6EndpointAddress 192.168.1.79:37286 (locally 192.168.1.79%lo)>>
DEBUG:coap-server:Socket error received, details: SockExtendedErr(ee_errno=111, ee_origin=2, ee_type=3, ee_code=3,
ee_pad=0, ee_info=0, ee_data=0)
DEBUG:coap-server:Incoming error 111 from <UDP6EndpointAddress 192.168.1.79:37286 (locally 192.168.1.79%lo)>
DEBUG:coap-server:Incoming message <aiocoap.Message at 0x7f05d622deb8: Type.CON POST (MID 1, empty token) remote
<UDP6EndpointAddress 192.168.1.79:37286 (locally 192.168.1.79%lo)>, 1 option(s), 4 byte(s) payload>
INFO:coap-server:Duplicate CON received, sending old response again
DEBUG:coap-server:Sending message <aiocoap.Message at 0x7f05d622df98: Type.ACK 2.04 Changed (MID 1, empty token)
remote <UDP6EndpointAddress 192.168.1.79:37286 (locally 192.168.1.79%lo)>>
DEBUG:coap-server:Socket error received, details: SockExtendedErr(ee_errno=111, ee_origin=2, ee_type=3, ee_code=3,
ee_pad=0, ee_info=0, ee_data=0)
DEBUG:coap-server:Incoming error 111 from <UDP6EndpointAddress 192.168.1.79:37286 (locally 192.168.1.79%lo)>
```

On peut observer plusieurs phénomènes :

- le serveur avait gardé en mémoire les requêtes précédentes mais ne les avait pas traitées car le programme avait été stoppé. En le redémarrant, il va pouvoir les traiter. Il répond donc au client qui peut afficher la notification 2.04 ;
- les autres requêtes sont vues comme des répétitions de la première puisque le champ Message ID est identique. Le serveur se contente de retourner une copie ;
- le client ayant terminé sa transaction a fermé la socket conduisant à l’émission d’un message ICMP qui conduit à l’affichage du message de log d’erreur (Socket error received).

Ceci peut être vérifié sur la capture Wireshark (cf. figure 13.5).

13.3.2 Ressource codée en CBOR

Le programme précédent, est correct mais nous avons utilisé le format ASCII de transfert par défaut (**Content-format = 0**). Il serait préférable de revenir au format JSON ou **CBOR** que nous avions vu précédemment. Pour ce faire, nous devons ajouter dans l’en-tête de la requête l’option Content-format. Comme son code est 12, elle doit être placée après l’option Uri-path (cf. tableau 12.3 page 161).

Listing 13.9 – coap_post_temp2.py

```
import CoAP
```

3. Le standard recommande d’ajouter un aléa lors de la transmission pour éviter que deux équipements se synchronisent et se perturbent toujours en même temps. Il n’est pas mis en œuvre dans cet exemple.

```

2 import socket
3 import time
4 try:
5     import kpn_senml.cbor_encoder as cbor #pycom
6 except:
7     import cbor2 as cbor # terminal on computer
8
9 SERVER = "192.168.1.26" # change to your server's IP address
10 PORT = 5683
11 destination = (SERVER, PORT)
12
13 s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
14
15 coap = CoAP.Message()
16 coap.new_header(code=CoAP.POST)
17 coap.add_option(CoAP.Uri_path, "temp")
18 coap.add_option(CoAP.Content_format, CoAP.Content_format_CBOR)
19 coap.add_payload(cbor.dumps(23.5))
20 coap.dump()
21
22 answer = CoAP.send_ack(s, destination, coap)
23 answer.dump()

```

Notez que ce programme, pprogcoap_post_temp2.py, peut aussi bien s'exécuter sur votre ordinateur ou sur votre LoPy. Dans le premier cas, le module `cbor2` sera utilisé. Sur le LoPy, nous ferons appel au module `kpn_senml`. Dans les deux cas, le contenu est transformé en CBOR grâce à la fonction `dumps`.

Du côté du serveur, la méthode `render_post` doit connaître le format de la ressource. Pour ce faire, elle doit accéder aux options CoAP.

Listing 13.10 – coap_basic_server3.py

```

34     return aiocoap.Message(payload=payload)
35
36 class TemperatureResource(resource.Resource):
37     async def render_post(self, request):
38         print ("-"*20)
39
40         ct = request.opt.content_format or \
41             aiocoap.numbers.media_types_rev['text/plain']
42
43         if ct == aiocoap.numbers.media_types_rev['text/plain']:
44             print ("text:", request.payload)
45         elif ct == aiocoap.numbers.media_types_rev['application/cbor']:
46             print ("cbor:", cbor.loads(request.payload))
47         else:
48             print ("Unknown format")
49             return aiocoap.Message(code=aiocoap.UNSUPPORTED_MEDIA_TYPE)
50     return aiocoap.Message(code=aiocoap.CHANGED)

```

Dans le programme `coap_basic_server3.py`, la variable `ct` contient la valeur de l'option **Content-format** si elle existe. De manière générale, `aiocoap` permet d'accéder aux valeurs de toutes les options CoAP contenues dans la requête. La valeur `None` indique que l'option n'est pas présente dans l'en-tête. Dans ce cas, la variable `ct` contiendra la valeur par défaut indiquant un texte en

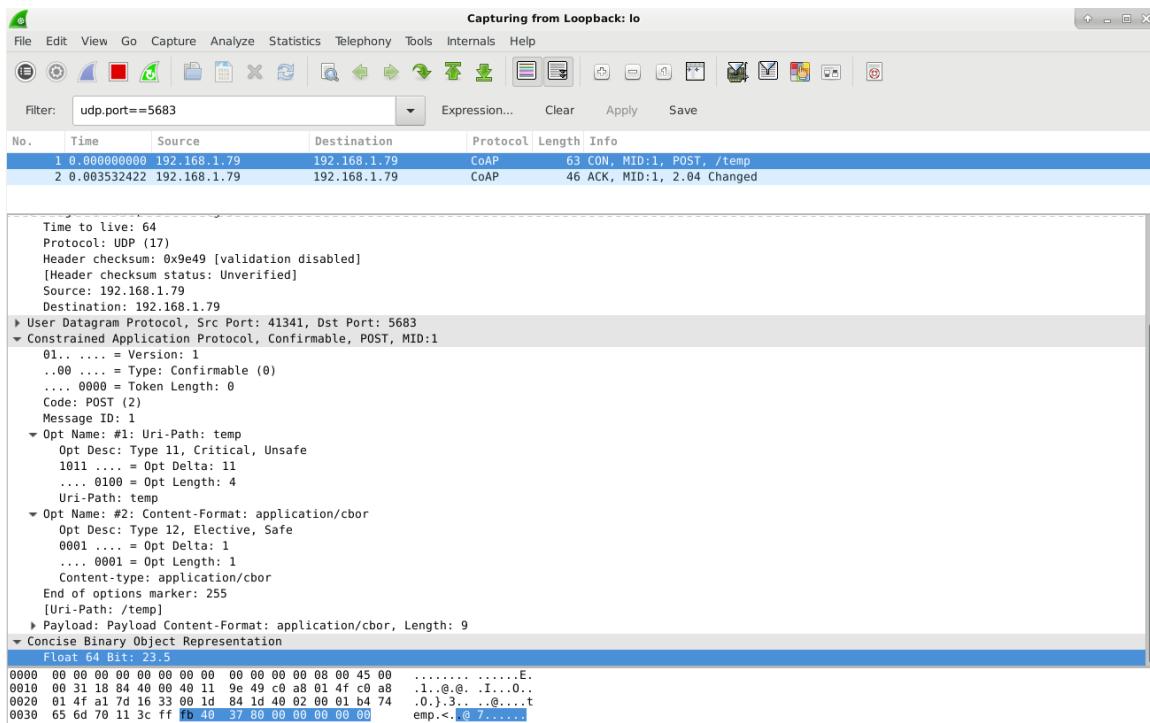


FIGURE 13.6 – Trafic complet lié au POST

ASCII.

Suivant le format de la donnée, le programme affichera le texte ou le CBOR transformé en chaîne de caractères avec la fonction `loads`. Si le format n'est pas connu du programme, un code d'erreur est retourné au client.

La figure 13.6 montre l'échange lié au POST et détaille contenu de la requête.

Question 13.3.1: Réponse

Que reçoit-on en réponse à la requête POST ?

- un message ACK
- le statut 2.00 OK.
- le statut 2.04 CHANGED.
- rien.

Question 13.3.2: JSON

Modifiez le programme client pour indiquer un content-format JSON. Quelle notification obtenez-vous ?

- un message RST
- le statut 4.04 NOT FOUND.
- le statut 2.15.
- le statut 5.00.

13.3.3 No Response

Que l'on utilise le type CONfirmable ou NON confirmable, le serveur va envoyer une notification :

- 2.xx quand tout se passe bien, et
- 4.xx ou 5.xx quand le client ou le serveur a commis une erreur.

Si un capteur veut utiliser CoAP sur un réseau LPWAN, à chaque POST qu'il va envoyer sur la voie montante (uplink), il va récupérer un acquittement dans la voie descendante (downlink). Or, on a vu que l'on devait ménager cette dernière qui pouvait être sujette à saturation ou à facturation.

L'option **No response** définie dans le RFC 7967 permet à un client d'informer le serveur qu'il ne souhaite pas recevoir de notifications lors d'un POST ou d'un PUT. La valeur de l'option est 258 et elle est suivie d'un octet contenant un bitmap qui va indiquer quel type de notification va être émis :

- si le deuxième bit à partir de la droite est mis à 1, le client ne veut pas recevoir les notifications de type 2.xx ;
- si le quatrième bit à partir de la droite est mis à 1, le client ne veut pas recevoir les notifications de type 4.xx ;
- si le cinquième bit à partir de la droite est mis à 1, le client ne veut pas recevoir les notifications de type 5.xx.

Par exemple, en mettant la valeur 0x02 dans cette option, le client ne reçoit pas les acquittements positifs mais uniquement les erreurs 4.xx et 5.xx. En mettant la valeur 0b00011010 (0x1a, 26), le serveur sera complètement silencieux.

Le programme `coap_post_temp4.py` ajoute cette option et le type du message CoAP a été fixé à NON confirmable. Cependant, le chemin d'URI n'est pas connu du serveur. On voit que le serveur retourne un message d'erreur. Si en revanche le POST se déroule correctement, seul le client émet une donnée et le serveur reste silencieux.

Listing 13.11 – `coap_post_temp4.py`

```

15 coap = CoAP.Message()
16 coap.new_header(type=CoAP.NON, code=CoAP.POST)
17 coap.add_option(CoAP.Uri_path, "temp")
18 coap.add_option(CoAP.Content_format, CoAP.Content_format_CBOR)
19 coap.add_option(CoAP.No_Response, 0b00000010)
20 coap.add_payload(cbor.dumps(23.5))
21 coap.dump()
```

13.4 Chaîne complète de remonté de mesures

Voilà ! nous pouvons mettre les différents concepts ensemble pour construire une chaîne complète de remontée d'informations sur la température, l'humidité et la pression.

Côté capteur, nous allons :

- si le BME280 est présent, récupérer directement les données. Sinon, nous allons utiliser les mesures virtuelles. Nous allons également définir le protocole réseau : WiFi, Sigfox, LoRaWAN ;
- si nous utilisons un réseau LoRaWAN, nous devons mettre en place un programme relais pour transmettre les données vers le serveur CoAP ;

Youtube



Le serveur CoAP doit pouvoir envoyer les données au serveur Beebotte pour affichage.

Le message CoAP va être configuré de la manière suivante :

- type = NON pour éviter les acquittements des messages CoAP ; type = NON to avoid acknowledgements of CoAP messages ;
- l’option No Response à 2 pour éviter les notifications REST positives. On garde sur la voie descendante les notifications d’erreurs pour permettre au capteur d’arrêter de transmettre ou d’augmenter sa période d’émission si le serveur n’est pas capable de traiter les données. Cela permettra de limiter l’usage du spectre et de préserver l’énergie des capteurs ; the No Response option to 2 to avoid positive REST notifications. Error notifications are kept on the downstream channel to allow the sensor to stop transmitting or to increase its transmission period if the server is not able to process the data. This will limit the use of the spectrum and preserve the energy of the sensors ;
- 3 chemins d’URI pour les 3 valeurs mesurées ; 3 URI paths for the 3 measured values ;
- le codage en CBOR des séries temporelles. CBOR coding of time series.

Il reste un point à traiter car avec la limitation à 12 octets des trames **Sigfox**, il est impossible de transmettre des données si on utilise CoAP. En effet, l’en-tête CoAP va contenir :

- 4 octets pour l’en-tête obligatoire ;
- 0 octets de Token ;
- 2 octets au minimum pour l’option URI-path si on réduit à un caractère le chemin ; par exemple T, H, P pour représenter la température, l’humidité et la pression ;
- 2 octets pour l’option **Content-format** indispensable pour indiquer que l’on transporte du CBOR ;
- 3 octets pour l’option **No-response**, également indispensable pour indiquer que l’on ne veut surtout pas d’acquittement (Sigfox les limitant à 4 par jour).

Soit, au total 11 octets. Il n’en reste plus qu’un. Si la température dépasse les 23 °C, l’information ne sera pas transportable.

13.5 SCHC

Static Context Header Compression, défini dans le [RFC 8724](#), donne l’acronyme SCHC que l’on prononce Chic. SCHC propose un mécanisme de compression générique des en-têtes. SCHC se base sur un contexte commun à l’émetteur et au récepteur qui va permettre d’éliminer les informations connues dans le message et de ne transmettre que les données qui ne peuvent pas être prédéterminées.

Nous allons mettre en œuvre une version simplifiée. Dans le message CoAP, nous avons besoin du champ Message ID qui va servir à éliminer les doublons qui pourraient apparaître dans le réseau. Les serveurs CoAP gardent en mémoire les informations concernant les Messages ID pendant 5 minutes. Donc, la numérotation des messages doit permettre des périodes plus longues avant de réutiliser les mêmes valeurs. Nous allons également transporter 3 types de ressources : /temperature, /humidity et /pressure.

On peut donc construire manuellement un en-tête SCHC avec les champs suivants :

- 2 bits pour numérotter les règles de compression. Dans notre cas, nous n’utiliserons que la règle 00 ;

- 4 bits pour numérotter les messages ID, ce qui donne 15 valeurs possibles de 1 à 15. Au pire, il faudrait émettre plus de 3 trames par minutes pour que le serveur CoAP traite deux trames différentes comme des doublons ;
- 2 bits pour désigner les chemins d'URI ; 3 seront utilisés.

Cela fait appel à plusieurs techniques de compression définies par SCHC :

- ***not_sent*** : la valeur d'un champ n'est pas envoyée sur le réseau car elle se trouve dans les règles. Cela s'appliquera à la plupart des champs ;
- ***Least Significant Bit*** : on n'envoie que les bits de poids faible. Cela s'applique au champ Message ID duquel on ne va transmettre que les 4 bits de poids faible ;
- ***Matching_sent*** : au lieu d'envoyer la valeur, on va envoyer un index sur un tableau commun. Cela s'applique à Uri-path où l'on enverra 00 pour l'élément temperature, 01 pour l'élément pressure, et 10 pour élément humidity.

13.5.1 Emission côté client

La compression très simplifiée que nous allons effectuer se fera au moment de l'envoi des données pour Sigfox dans le programme `coap_full_sensor.py`.

Listing 13.12 – `coap_full_sensor.py`

```

174 def send_coap_message(sock, destination, uri_path, message):
175     if destination[0] == "SIGFOX": # do SCHC compression
176         global sigfox_MID
177
178         """ SCHC compression for Sigfox, use rule ID 0 stored on 2 bits,
179         followed by MID on 4 bits and 2 bits for an index on Uri-path.
180
181         the SCHC header is RRMMMMUU
182         """
183
184         uri_idx = ['temperature', 'pressure', 'humidity', 'memory'].index(uri_path)
185
186         schc_residue = 0x00 # ruleID in 2 bits RR
187         schc_residue |= (sigfox_MID << 2) | uri_idx # MMMM and UU
188
189         sigfox_MID += 1
190         sigfox_MID &= 0x0F # on 4 bits
191         if sigfox_MID == 0: sigfox_MID = 1 # never use MID = 0
192
193         msg = struct.pack("!B", schc_residue) # add SCHC header to the message
194         msg += cbor.dumps(message)
195
196         print ("length", len(msg), binascii.hexlify(msg))
197
198         s.send(msg)
199         return None # don't use downlink

```

Dans le cas de Sigfox, on prend l'index en cherchant l'élément dans le tableau (ligne 183). On construit ensuite l'octet SCHC en ajoutant un numéro de règle (ligne 185) et les 4 bits de poids faible du champ Message ID qui va donc varier de 1 à 15 (ligne 186). Puis, l'on concatène les données CBOR à envoyer (ligne 193).

Pour les autres technologies de transmission, l'en-tête CoAP est construite avec les fonctions du module CoAP.py.

```

200     # for other technologies we send a regular CoAP message
201     coap = CoAP.Message()
202     coap.new_header(type=CoAP.NON, code=CoAP.POST)
203     coap.add_option(CoAP.Uri_path, uri_path)
204     coap.add_option(CoAP.Content_format, CoAP.Content_format_CBOR)
205     coap.add_option(CoAP.No_Response, 0b00000010) # block 2.xx notification
206     coap.add_payload(cbor.dumps(message))
207     coap.dump(hexa=True)
208     answer = CoAP.send_ack(s, destination, coap)

```

L'envoi de la trame se fait avec la fonction `send_ack`. Comme le message est de type NON, cette fonction n'attend pas de réponse après l'émission des données.

13.5.2 Réception côté serveur

Si les données émises par `coap_full_sensor.py` passent par un réseau LPWAN, le programme `generic_coap_relay.py` va servir d'intermédiaire pour les envoyer au serveur CoAP. Le programme est presque identique à celui que nous avions utilisé pour transmettre dans la session précédente. Les seules différences sont :

- les numéros de ports utilisés : 5683 au lieu de 33033 ;
- l'ajout de la décompression SCHC pour les données venant de Sigfox.

Listing 13.13 – generic_coap_relay.py

```

# Sigfox use SCHC compression, first byte is CoAP compressed header
70    SCHC_byte = payload[0]

72    ruleID = SCHC_byte >> 6
73    mID = (SCHC_byte & 0b00111100) >> 2
74    uri_idx = SCHC_byte & 0b0000_0011

76    m = aiocoap.message.Message(
77        mtype=NON,
78        code=POST,
79        mid=mID,
80        payload=payload[1:])
81
82    m.opt.uri_path = (
83        ["temperature", "pressure", "humidity", "memory"][uri_idx],
84    )

86    m.opt.content_format = 60
87    m.opt.no_response = 0b0000_0010
88
89    downlink = forward_data(m.encode())

```

Pour reconstruire l'en-tête CoAP, SCHC se base sur des règles pour rendre le traitement indépendant des champs employés par le protocole. Mais ici, pour faire plus simple, nous utilisons le module Message d'`aiocoap` pour reconstituer le message CoAP.

Dans un premier temps, on extrait du premier octet la valeur du champ message ID et l'index des URI (ligne 72 à 74) puis, à partir de ces valeurs et de celles connues à l'avance, le message CoAP est reconstitué.

Dans tous les cas, la fonction `forward_data` est utilisée. Elle retourne la réponse du serveur CoAP qui est renvoyée au réseau LPWAN suivant les principes que nous avions vus lors de la session précédente. Nous ne l'avons pas mis en œuvre pour Sigfox pour éviter une erreur vu que 4 messages par jours sont autorisés.

13.5.3 serveur CoAP

Finalement, le programme `coap_server.py` a été étendu à différents chemins d'URI pour traiter les différentes ressources que vont nous envoyer les capteurs. Et dans le traitement de la ressource, l'appel à la fonction `to_bbt` a été ajouté.

13.6 Pistes d'améliorations

Vous pouvez étendre cet ensemble de programmes. Voici quelques pistes d'amélioration :

- les capteurs envoient également leur mémoire disponible. Cela peut être utile pour détecter une fuite dans la mémoire ; par exemple quand une structure n'est jamais libérée. La structure CBOR est bien envoyée mais ni `coap_server.py` ni Beebotte n'ont été configurés pour afficher ces valeurs. Vous pouvez donc l'intégrer dans la chaîne de traitement ;
- le POST de la mémoire conduit à l'émission d'un message en downlink pour indiquer l'erreur 4.04. C'est une conséquence de l'utilisation de l'option CoAP `no_response` qui ne bloque que les notifications de type 2.xx. Vous pouvez modifier le module CoAP.py pour que la réponse soit traitée. Par exemple, en limitant à un envoi par jour de cette ressource ;
- en plus de la mémoire, il peut être intéressant d'envoyer le niveau de la batterie. La page [Correct formula for BATT monitoring on expansion board | Pycom user forum](#)⁴ donne des indications pour récupérer le niveau de la batterie ;
- finalement, nous n'avons pas réglé tous les problèmes d'interopérabilité. Si, par exemple, vous voulez envoyer toutes les heures un relevé de la mémoire libre et toutes les minutes la température, vous devez également modifier le programme `coap_server.py`. Si ces paramètres sont transmis de manière optimale par le serveur, le programme `coap_server` peut devenir complètement indépendant de la valeur mesurée.

4. <https://forum.pycom.io/topic/1690/correct-formula-for-batt-monitoring-on-expansion-board>



14. LwM2M

Il est temps de voir une vrai plateforme IoT qui met en œuvre ce que nous venons d'apprendre sur CoAP et REST. Il faut installer deux programmes **java**¹ :

```
wget https://ci.eclipse.org/leshan/job/leshan/lastSuccessfulBuild/artifact/leshan-client-demo.jar
```

et

```
wget https://ci.eclipse.org/leshan/job/leshan/lastSuccessfulBuild/artifact/leshan-server-demo.jar
```

14.1 Introduction

Lightweight Machine to Machine (LwM2M) est une architecture définie par l'Open Mobile Alliance (OMA) à l'origine pour permettre aux opérateurs de gérer certaines ressources sur les téléphones mobiles. Mais cette architecture peut être étendu à d'autres environnement.

Les spécifications sont accessibles sur le site de l'OMA². Nous allons utiliser une mise en œuvre en java appelé . Pas de panique nous n'allons pas programmer nous aurons juste besoin d'un navigateur Web et de **Wireshark**.

LwM2M comme son nom l'indique se veut léger, c'est-à-dire que les mises en œuvre ne doivent pas être trop complexes et que le trafic engendré ne doit pas non plus être trop important. LwM2M est une plate-forme et elle va donc faire plus qu'un simple trafic REST. En particulier, l'un des rôles est de permettre aux objets de s'enregistrer et de décrire leurs caractéristiques. LwM2M va également structurer fortement les ressources en imposant des règles de nomsages relativement contraintes et le format des ressources.

1. Il s'agit d'une mise en œuvre assez ancienne, de nouvelles spécifications existent et sont quelque peu différentes, mais les concepts n'ont pas changés

2. <https://omaspecworks.org/what-is-oma-specworks/iot/lightweight-m2m-lwm2m/>

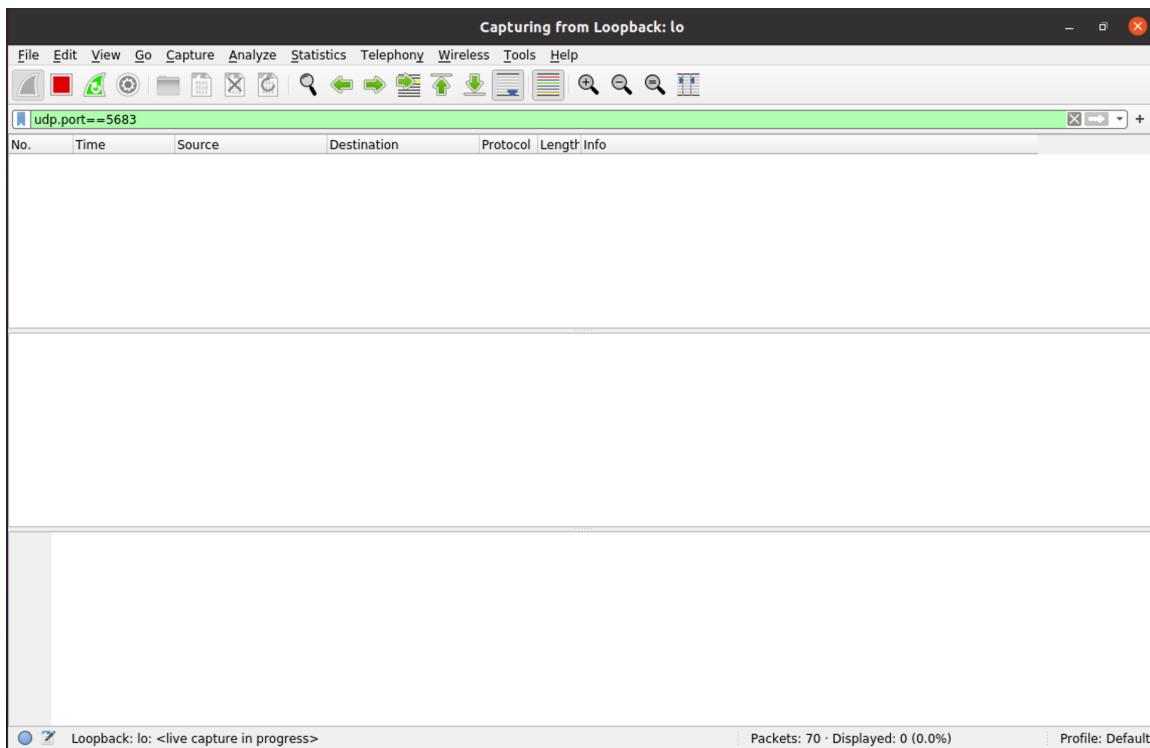


FIGURE 14.1 – Initialisation de Wireshark pour capturer le trafic CoAP

14.2 Architecture

Comme tout système qui se respecte, LwM2M fonctionne en mode client/serveur. Le serveur est la plate-forme de gestion des objets et les clients sont les objets connectés sur le réseau.

Dans un premier temps, lancez Wireshark sur l'interface *loopback* et filtrez le trafic en le limitant au port 5683 pour UDP (celui de CoAP) en tapant `udp.port==5683`. Comme le montre la figure 14.1.

Nous allons maintenant lancer le serveur LwM2M, tapez³ :

```
> java -jar ./leshan-server-demo.jar
2020-10-22 01:34:09,365 INFO LeshanServer - LWM2M server started at
coap://0.0.0.0/0.0.0.0:5683 coaps://0.0.0.0/0.0.0.0:5684
2020-10-22 01:34:09,553 INFO LeshanServerDemo - Web server started at
http://0.0.0.0:8080/.
```

Il nous indique qu'il utilise le port 5683 pour CoAP et que l'on peut superviser le serveur avec un navigateur sur le port 8080.

Lancez le navigateur sur l'URI `http://127.0.0.1:8080`, la page indiquée figure 14.2 page suivante apparaît.

Vous pouvez remarquer que l'ouverture du serveur LwM2M n'a provoqué aucun trafic CoAP sur l'analyseur réseau.

3. Sous Linux, tapez `apt install -y default-jre` pour installer Java.

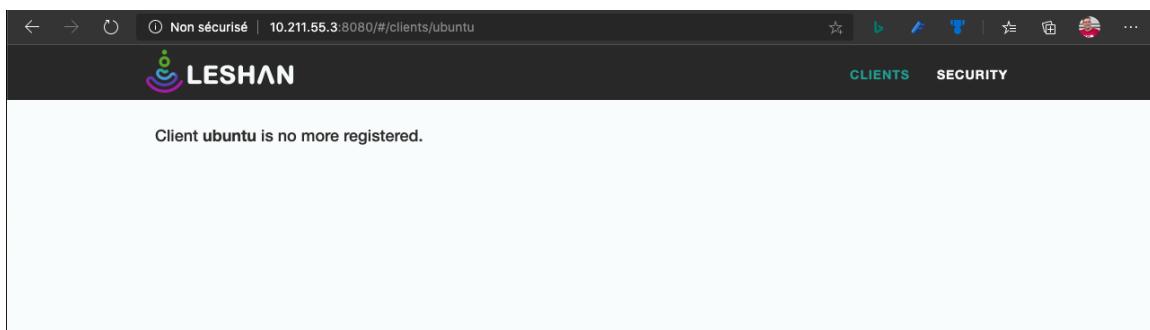


FIGURE 14.2 – Page d'accueil du serveur LwM2M

Lancez maintenant dans une autre fenêtre, le client LwM2M :

```
> java -jar ./leshan-client-demo.jar
2020-10-22 01:49:51,063 INFO LeshanClientDemo - Commands available :

- create <objectId> : to enable a new object.
- delete <objectId> : to disable a new object.
- update : to trigger a registration update.
- w : to move to North.
- a : to move to East.
- s : to move to South.
- d : to move to West.

2020-10-22 01:49:51,064 INFO LeshanClient - Starting Leshan client ...
2020-10-22 01:49:51,158 INFO CaliforniumEndpointsManager - New
endpoint created for server coap://localhost:5683 at
coap://0.0.0.0:48274
2020-10-22 01:49:51,159 INFO LeshanClient - Leshan client
[endpoint:ubuntu] started.
2020-10-22 01:49:51,160 INFO DefaultRegistrationEngine - Trying to
register to coap://localhost:5683 ...
2020-10-22 01:49:51,252 INFO DefaultRegistrationEngine - Registered
with location '/rd/BOr5Pg7yW8'.
2020-10-22 01:49:51,252 INFO DefaultRegistrationEngine - Next
registration update to coap://localhost:5683 in 53s...
```

14.3 Enregistrement d'un Objet

Comme nous utilisons une adresse *loopback*, il est un peu plus difficile de repérer dans le trafic Wireshark récupéré (cf. figure 14.3 page suivante) qui est le client et qui est le serveur. Mais comme Le serveur attend sur le port 5683, en regardant de plus près un paquet, on peut déterminer s'il a été émis par le client ou le serveur.

14.3.1 Analyse de l'en-tête CoAP

On peut voir sur la trace, que le client contacte le serveur pour lui indiquer ses propriétés.

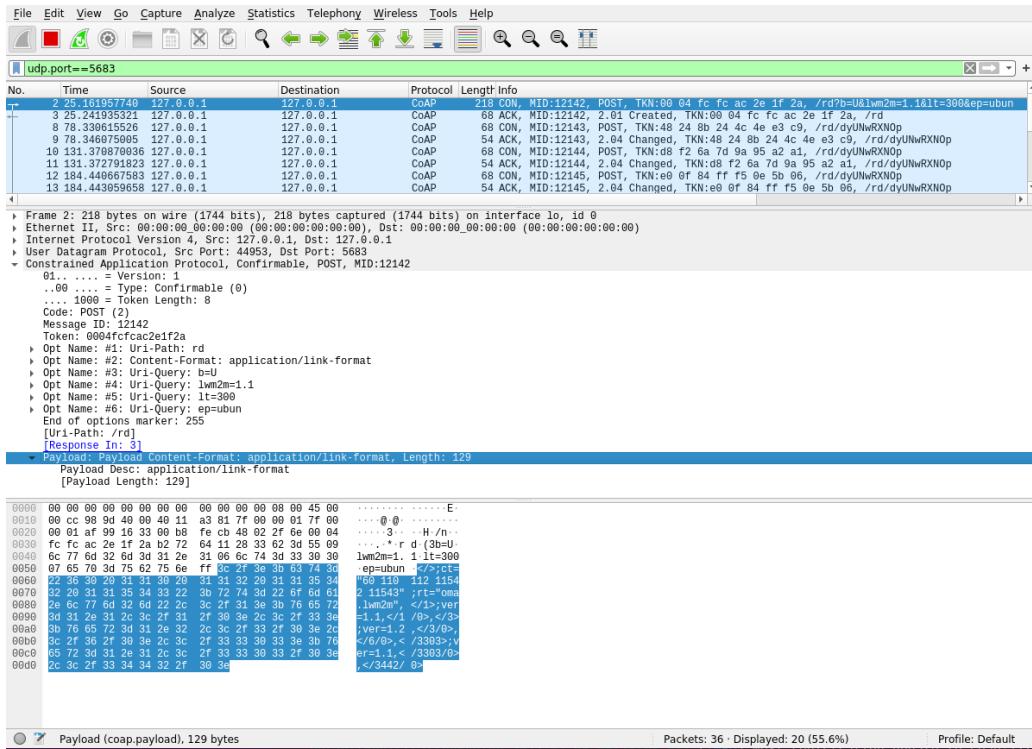


FIGURE 14.3 – Premières captures LwM2M

Question 14.3.1: Méthode

Quelle est la nature de la requête émise par le client :

- GET
- POST
- PUT
- DELETE

Question 14.3.2: URI

Quelle est le chemin de l'URI :

- vide
- /rd
- /lwm2m

Question 14.3.3: Content

Quelle est le format du contenu (content-format) :

- text
- XML
- JSON
- link-format
- CBOR

Question 14.3.4: Période

A quelle période voyez-vous d'afficher des messages CoAP ?

Pour décoder l'Uri-Query de la requête CoAP il faut d'aider du document LwM2M Core Specification⁴ et de son tableau 6.2.1 page 40.

Question 14.3.5: b=U

Que signifie b=U ?

- Les données seront émises avec les unités (Units)
- Le référentiel des unités est la représentation Universelle
- Le protocole sous-jacent est en mode datagramme (UDP)
- Le client n'est pas référencé (Unreferenced)

Question 14.3.6: lwm2m=1.1

Que signifie lwm2m=1.1 ? Il s'agit...

- de la version lwm2m du client
- de la taille mémoire de l'implémentation (1.1 ko)
- de la version lwm2m du serveur

Question 14.3.7: lt=300

Que signifie lt=300 ?

- C'est le nombre maximal d'objets (less than 300)
- C'est la taille maximale d'un échange (300 octets)
- C'est la durée de vie de l'enregistrement d'un objet (lifetime)

14.3.2 Analyse du contenu du POST

Revenons au contenu du message de la requête POST émise initialement par le client. Que veut dire **link-format**? Nous n'avons pas encore vu ce format. Heureusement, l'IANA est notre ami et en allant chercher à quoi correspond cette valeur, sur cette page⁵, on trouve que le [RFC 6690](#) définit ce contenu.

4. http://www.openmobilealliance.org/release/LightweightM2M/V1_1_1-20190617-A/OMA-TS-LightweightM2M_Core-V1_1_1-20190617-A.pdf

5. <https://www.iana.org/assignments/core-parameters/core-parameters.xhtml#content-formats>

Il utilise un format particulier qui est utilisé initialement par le Web pour définir des relations entre pages initialement définies dans le [RFC 5988](#). Il est important de remarquer, et après la lecture devient plus claire, que les URI sont notées entre <>. Ensuite, on trouve des attributs liés à cet URI séparés par des points-virgules. Les virgules séparent les définitions.

En suivant ces règles de notation, les données émises par le client peuvent être formatés de la manière suivante :

```
</>;ct="60 110 112 11542 11543";rt="oma.lwm2m",
</1>;ver=1.1,
</1/0>,
</3>;ver=1.2,
</3/0>,
</6/0>,
</3303>;ver=1.1,
</3303/0>,
</3442/0>
```

Le client utilise ce format pour décrire les 9 ressources qu'il possède et qui sont identifiées par ces 9 chemins d'URI. Le nommage des ressources peut paraître étrange ; nous verrons par la suite à quoi il correspond :

- La première ligne concerne la racine </> pour laquelle deux attributs sont associés, il seront donc applicable à l'ensemble des éléments. La première ligne concerne la racine </> pour laquelle deux attributs sont associés, il seront donc applicable à l'ensemble des éléments.
- Le premier `ct`⁶ définit les formats des représentations possibles des objets. La partie entre guillemets fait référence aux valeurs de Content-format de CoAP listé tableau 12.4 page 163. On y retrouve respectivement les types CBOR, SenML et pour les deux derniers un format orienté TLV propre à LwM2M.
- Le second `rt`⁷ indique le type de ressource (*resource-type*), c'est-à-dire indique comment elles seront représentées. Ici, cela indique que les ressources suivront les spécifications LwM2M de l'OMA.
- Les lignes suivantes décrivent, toujours de manière hiérarchique, les chemins d'URI et les attributs associés. L'attribut `ver` indique la version du standard utilisée pour définir la ressource.

Question 14.3.8: rt

À quoi sert l'attribut `rt` ?

- à donner la sémantique (i.e. comment interpréter) de la ressource.
- à indiquer la taille de la ressource en faisant référence à lwm2m.
- à indiquer la version de lwm2m utilisée.
- à aider à déboguer les échanges.

14.3.3 Définition des ressources

LwM2M représente les ressources d'une manière assez originale pour être à la fois compact, universel (ce qui est quelquefois oxymorique) et flexible. Pour ce faire, tout est désigné par des

6. Voir [RFC 7252](#) chapitre 7.2.

7. Voir [RFC 6690](#) chapitre 3.1.

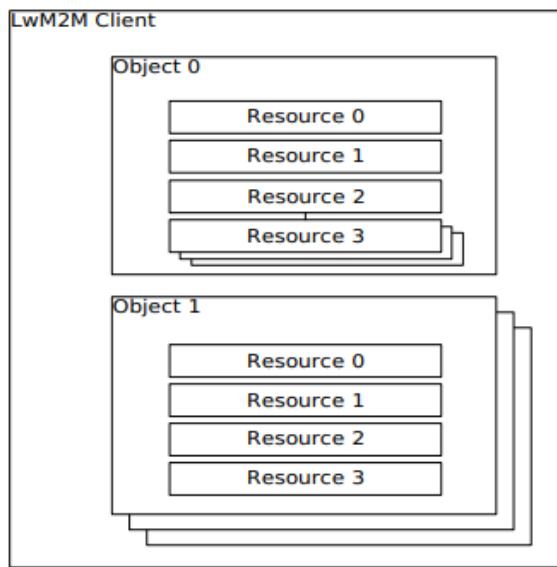


Figure: 7.1.-1 Relationship between LwM2M Client, Object, and Resources

FIGURE 14.4 – Hierarchie de nommage

chiffres. Le chapitre 7 page 63 du document principal⁸ contient le schéma figure 14.4 illustrant cette hiérarchie :

- Un Objet physique (ie notre client) contient une liste d'objets numériques⁹ identifié par un numéro. Ce numéro est attribué par l'OMA. Par exemple, un capteur de température aura la valeur 3303. Une liste des valeurs déjà attribuées peut être trouvée à cette adresse¹⁰.
- Un client peut évidemment posséder plusieurs instance d'un objet numérique, par exemple, une station météo pourrait avoir un capteurs de température intérieur et extérieur. Le deuxième chiffre du chemin de l'URI indique cette instance. S'il n'y en a qu'un, il prend la valeur 0.
- Finalement, l'objet peut être complexe et contenir plusieurs informations appelée ressources. En cliquant sur le chiffre 3303 dans la page web précédente, on obtient la description de l'objet. On peut voir que :
 - 5700 représente la valeur mesurée par le capteur
 - 5601 la valeur minimale
 - 5602 la valeur maximale
 - 5701 indique les unités
 - 5605 permet de remettre à zéro le calcul des minima et maxima
 Ces valeurs peuvent se retrouver dans plusieurs objets numériques.
- la version 1.2 du standard introduit également la possibilité d'avoir plusieurs instances d'une ressource.

8. http://www.openmobilealliance.org/release/LightweightM2M/V1_1_1-20190617-A/OMA-TS-LightweightM2M_Core-V1_1_1-20190617-A.pdf

9. LwM2M appelle cette information *object* ce qui rend en français la définition ambiguë puisque Objet est aussi la traduction de *thing*

10. <http://www.openmobilealliance.org/wp/OMNA/LwM2M/LwM2MRegistry.html#resources>

Valeurs associées aux objets numériques

Les objets numériques peuvent être définis par LwM2M, ils concernent ceux dédiés à la gestion du protocole. Par exemple, l'objet numérique :

- *0* définit l'environnement de sécurité du client. Il contient l'adresse du serveur, les clés de chiffrement,...
- *1* définit les paramètres pour la communication avec le serveur, comme par exemple la durée de vie de l'information en seconde,...
- *3* describes the characteristics of the equipment as its software version,...
- ...

Les valeurs comprises entre 2 048 et 10 240 sont définies par des organismes de standardisation partenaire de l'OMA, comme par exemple :

- l'IPSO Alliance qui va définir des objets numériques types, comme un capteur de température ayant pour référence *3303*,
- la GSMA qui définit les objets pour la gestion des téléphones cellulaires, the GSMA which defines the objects for the management of cell phones,
- ...

Les valeurs supérieures sont réservées aux industriels qui peuvent enregistrer leurs propres spécifications.

Valeurs associées aux ressources

La figure 14.5 page suivante, copiée du site Web de l'OMA, donne un exemple de définition d'un objet numérique. A l'objet *3303* sont associés un certain nombre de ressources, générique que l'on peut aussi trouver dans d'autres objets numériques. Ainsi :

- *5700* refers to the measured value represented as a floating number. This number can only be read (R)
- *5700* est une chaîne de caractère qui précise l'unité de mesure.
- *5601* et *5602* conservent les valeurs minimales et maximales. Elles peuvent être remises à zéro via la ressource *5605* qui conduit à une exécution d'un programme (E).
- *5603* et *5604* correspondent à la plage d'utilisation du capteur et ne peuvent pas être modifiées.

Ainsi *3303/3/5601* représente la valeur minimale (*5601*) du quatrième capteur (3 car on commence à 0) de l'objet numérique température (*3303*).

Par rapport à **Modbus**, où l'on devait télécharger la documentation de l'Objet (cf tableau 4.3 page 56), LwM2M offre une manière standard de décrire l'information produite ou consommée par un Objet. De plus, la définition de l'objet numérique, en plus d'être visualisable sous forme de tableau sur le site web, existe aussi en **XML** pour être traitée informatiquement et permettre l'interopérabilité.

```
<?xml version="1.0" encoding="UTF-8"?>
<! -- BSD-3 Clause License ... -->
<LWM2M xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="http://openmobilealliance.org/tech/profiles/LWM2M.xsd">
    <Object ObjectType="MODefinition">
        <Name>Temperature</Name>
        <Description1>
            This IPSO object should be used with a temperature sensor to report a temperature measurement. It also provides resources for minimum/maximum measured values and the minimum/maximum range that can be measured by the temperature sensor. An example measurement unit is degrees Celsius.
        </Description1>
        <ObjectID>3303</ObjectID>
        <ObjectURN>urn:oma:lwm2m:ext:3303:1.1</ObjectURN>
```

Temperature

Description

This IPSO object should be used with a temperature sensor to report a temperature measurement. It also provides resources for minimum/maximum measured values and the minimum/maximum range that can be measured by the temperature sensor. An example measurement unit is degrees Celsius.

Object definition

Name	Object ID	Object Version	LWM2M Version
Temperature	3303	1.1	1.0
Object URN		Instances	Mandatory
urn:oma:lwm2m:ext:3303:1.1		Multiple	Optional

Resource Definitions

ID	Name	Operations	Instances	Mandatory	Type	Range or Enumeration	Units	Description
5700	Sensor Value	R	Single	Mandatory	Float			Last or Current Measured Value from the Sensor.
5601	Min Measured Value	R	Single	Optional	Float			The minimum value measured by the sensor since power ON or reset.
5602	Max Measured Value	R	Single	Optional	Float			The maximum value measured by the sensor since power ON or reset.
5603	Min Range Value	R	Single	Optional	Float			The minimum value that can be measured by the sensor.
5604	Max Range Value	R	Single	Optional	Float			The maximum value that can be measured by the sensor.
5701	Sensor Units	R	Single	Optional	String			Measurement Units Definition.
5605	Reset Min and Max Measured Values	E	Single	Optional				Reset the Min and Max Measured Values to Current Value.
5750	Application Type	RW	Single	Optional	String			The application type of the sensor or actuator as a string depending on the use case.
5518	Timestamp	R	Single	Optional	Time			The timestamp of when the measurement was performed.
6050	Fractional Timestamp	R	Single	Optional	Float	0..1	s	Fractional part of the timestamp when sub-second precision is used (e.g., 0.23 for 230 ms).
6042	Measurement Quality Indicator	R	Single	Optional	Integer	0..23		Measurement quality indicator reported by a smart sensor. 0: UNCHECKED No quality checks were done because they do not exist or can not be applied. 1: REJECTED WITH CERTAINTY The measured value is invalid. 2: REJECTED WITH PROBABILITY The measured value is likely invalid. 3: ACCEPTED BUT SUSPICIOUS The measured value is likely OK. 4: ACCEPTED The measured value is OK. 5-15: Reserved for future extensions. 16-23: Vendor specific measurement quality.
6049	Measurement Quality Level	R	Single	Optional	Integer	0..100		Measurement quality level reported by a smart sensor. Quality level 100 means that the

FIGURE 14.5 – Définition de l'objet numérique 3303 pour la température

```

<LWM2MVersion>1.0</LWM2MVersion>
<ObjectVersion>1.1</ObjectVersion>
<MultipleInstances>Multiple</MultipleInstances>
<Mandatory>Optional</Mandatory>
<Resources>
    <Item ID="5700">
        <Name>Sensor Value</Name>
        <Operations>R</Operations>
        <MultipleInstances>Single</MultipleInstances>
        <Mandatory>Mandatory</Mandatory>
        <Type>Float</Type>
        <RangeEnumeration></RangeEnumeration>
        <Units></Units>
        <Description>Last or Current Measured Value from the Sensor.</Description>
    </Item>
    <Item ID="5601">
        <Name>Min Measured Value</Name>
        <Operations>R</Operations>
        <MultipleInstances>Single</MultipleInstances>
        <Mandatory>Optional</Mandatory>
        <Type>Float</Type>
        <RangeEnumeration></RangeEnumeration>
        <Units></Units>
        <Description>
            The minimum value measured by the sensor since power ON or reset.
        </Description>
    </Item>
...

```

Question 14.3.9: 3301/0/5602

En vous aidant de la page de description des objets numériques et des ressources^a, que représente l'URI *3301/0/5602* ?

- la température maximale du premier capteur
- l'humidité minimale du capteur 0
- la luminosité maximale du capteur 0

a. <http://www.openmobilealliance.org/wp/OMNA/LwM2M/LwM2MRegistry.html#resources>

Question 14.3.10: 10340/0/2

Que représente l'URI *10340/0/2* ?

- la température en Fahrenheit de l'objet 10340
- la longitude dans des coordonnées GPS
- le statut actif ou non d'une caméra

Question 14.3.11: Dimmer

Quelle URI permet d'accéder à l'élément contrôlant la variation lumineuse (*Dimmer*) d'un éclairage (*Light Control*) ? Quelle valeur maximale peut prendre cette ressource ?

Question 14.3.12: Hz

Quel serait le chemin d'URI pour obtenir la fréquence d'un courant électrique sur la deuxième phase ?

Question 14.3.13: Qui est qui ?

Dans le premier échange capturé :

```
</>;ct="60 110 112 11542 11543";rt="oma.lwm2m",
</1>;ver=1.1,
</1/0>,
</3>;ver=1.2,
</3/0>,
</6/0>,
</3303>;ver=1.1,
</3303/0>,
</3442/0>
```

1

Location

3

Device

6

LwM2M v1.1 Test Object

3303

LwM2M Server

3442

Temperature

14.4 Resource Directory

Nous allons nous focaliser sur le premier échange que nous avons commencé à analyser en regardant le message émis par le client vers le serveur LwM2M. Comme nous l'avons vu, ce premier message contient la description des ressources présentes sur le client. Les valeurs des identifiants d'objets numérique (Object ID) et de ressources (Resource ID) permettent au serveur de savoir ce que peut faire le client, vu qu'elles sont normalisées. Le client LwM2M envoie ces informations sur un chemin d'URI bien connu /rd (pour *resource description*).

Question 14.4.1: Réponse du serveur LwM2M

Que répond le serveur (cf. figure 14.6 page suivante) ?

- Il acquitte juste le message.
- Rien. Nothing.
- Il renvoie une URI qui servira par la suite à identifier l'objet. It returns a URI that will be used later to identify the object.
- Il retourne les données de chiffrement des messages. It returns the message encryption data.

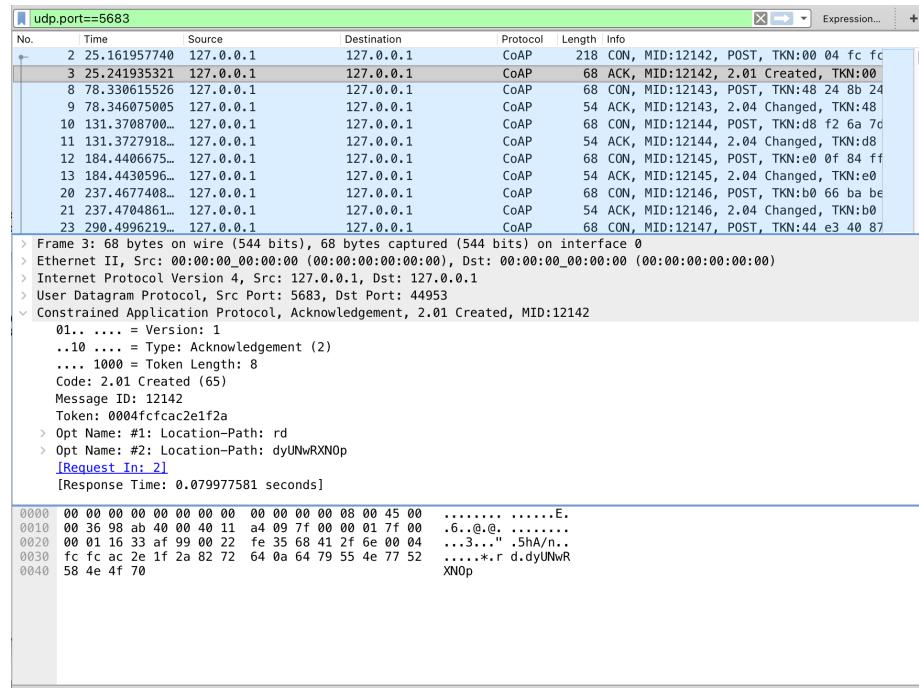


FIGURE 14.6 – Réponse du serveur LwM2M au POST du client

Question 14.4.2: Wait and See

Si on laisse la plate-forme fonctionner sans intervenir, que voit-on sur l'analyseur réseau ?

- Rien.
- Le capteur envoie des informations indiquant un changement de valeur mesurée.
- Le client envoie les valeurs mesurées même s'il elles n'ont pas changé.
- Le client envoie un message vide vers le serveur pour indiquer qu'il est toujours présent.
- Le serveur envoie un message vide vers ses clients pour savoir s'ils sont toujours présents.

14.5 Interrogation du client LwM2M

Il est temps de revenir à l'interface de la plate-forme en ouvrant la page `http://127.0.0.1:8080` depuis le navigateur. Assurez vous que Wireshark capture toujours le trafic sur l'interface *loopback*. L'objet que nous avons inscrit apparaît. Cliquez dessus. Vous devez avoir quelque chose comme ce qui est représenté figure 14.7 page ci-contre.

Dans le menu de gauche, on retrouve les objets numériques LwM2M décrit lors du premier POST de l'Objet.

En cliquant sur *Temperature*, les ressources LwM2M sont affichées. Les petits logos permettent d'effectuer des actions :

- *R* pour lire une ressource, *W* pour l'écrire et *EXE* pour exécuter un code (l'engrenage permet de définir les paramètres) ;

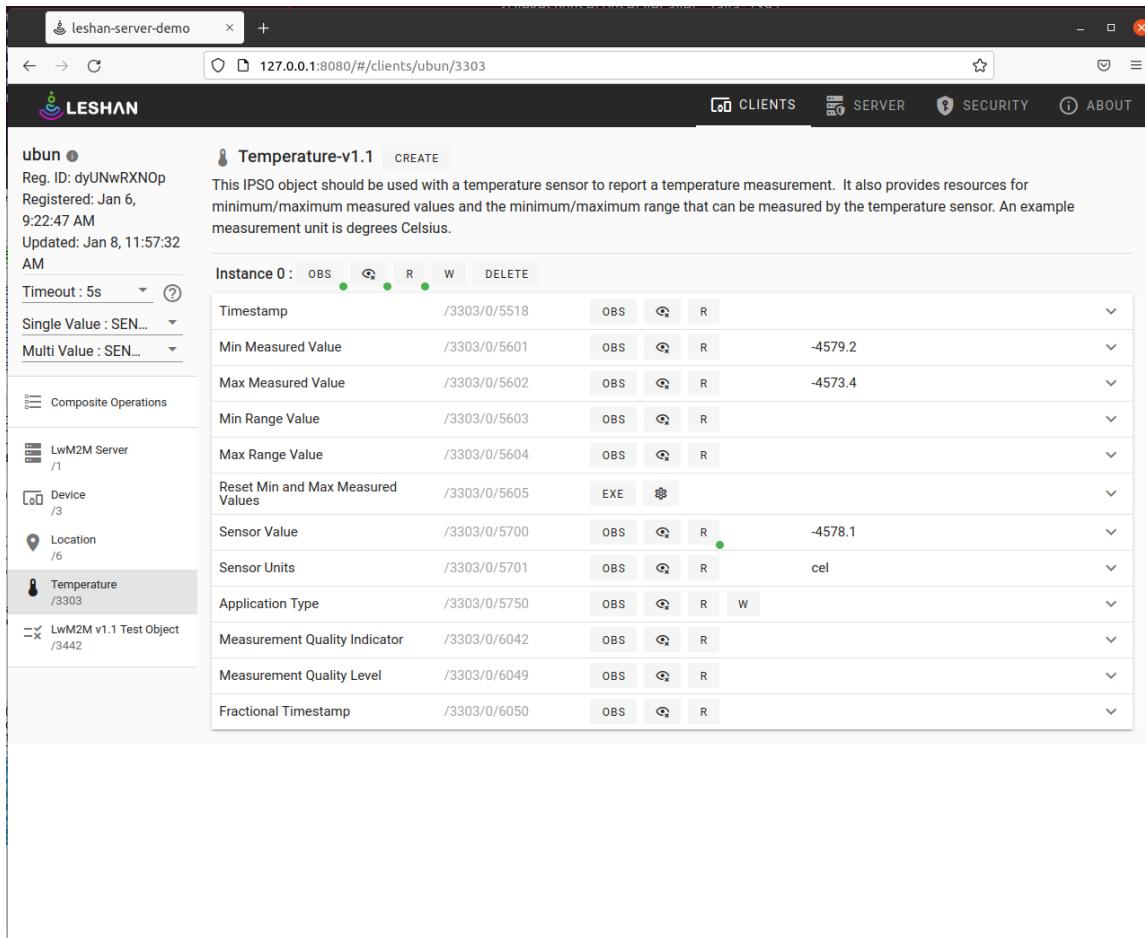


FIGURE 14.7 – Visualisation du serveur LwM2M

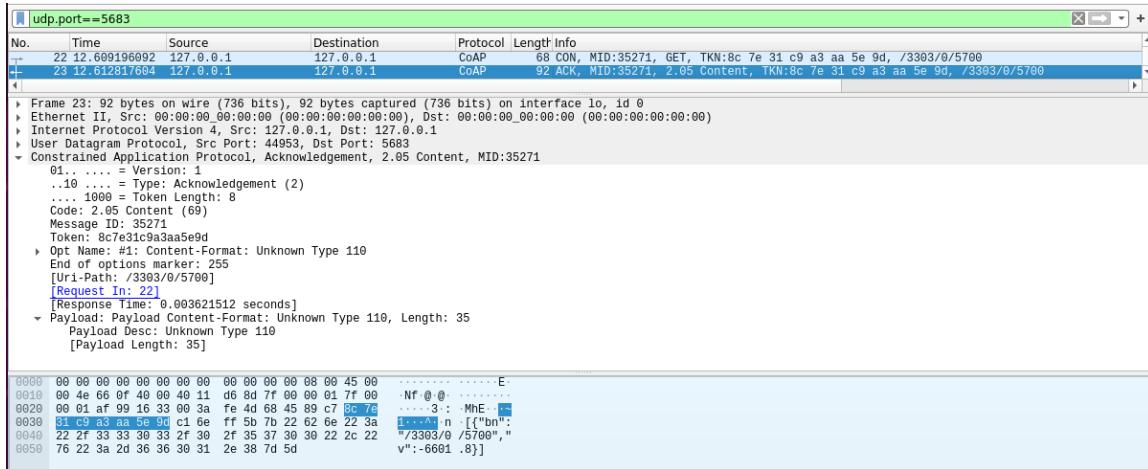


FIGURE 14.8 – Interrogation d'une ressource

— *OBS* permet de lancer un **Observe** sur une ressource, L'oeil avec la croix permet d'annuler un **Observe**.

Ces actions peuvent s'appliquer individuellement à chaque ressource, ou plus globalement à une instance d'un objet numérique.

14.5.1 Lecture simple

Dans le menu du gauche, sélectionnez SENML_JSON dans le menu déroulant *Single Value*. Cela permettra d'utiliser le format **SenML** plutôt que celui spécifié par LwM2M basé sur des TLV.

Pour l'objet numérique *Temperature* cliquez sur le bouton *R* de *Sensor Value*. La figure 14.8 monte cet échange et détaille la réponse. Le Serveur LwM2M agit comme un client REST et envoie au serveur REST du client LwM2M la requête :

GET /3303/0/5700

en indiquant le type 110 dans l'option **Accept** (cf. tableau 12.4 page 163) qui correspond au format **SenML** codé en JSON .

La réponse doublement associée par le même **Token** et un message de type **ACK** de format SenML en JSON :

[{"bn": "/3303/0/5700", "v": -6601.8}]

On y retrouve le nom de base (bn) correspondant au nom de la ressource et la valeur (v)¹¹.

Cette valeur s'affiche dans la page web du serveur LwM2M, le petit point vert indiquant que l'échange s'est déroulé correctement (cf. figure 14.9 page suivante).

¹¹. Il est évident que le résultat retourné n'a aucun sens physique, le client LwM2M émulant mal l'évolution d'une température sur le long terme.

Resource ID and Max Measurement Values	/3303/0/5605	EXE			v
Sensor Value	/3303/0/5700	OBS		R	-6601.8
Sensor Units	/3303/0/5701	OBS		R	

FIGURE 14.9 – Relevé de la température

117.992509491 127.0.0.1	127.0.0.1	CoAP	96 NON, MID:17275, 2.05 Content, TKN:ff 6a 04 4d 3a 24 60 6d, /3303/0/5700
119.992474211 127.0.0.1	127.0.0.1	CoAP	96 NON, MID:17276, 2.05 Content, TKN:ff 6a 04 4d 3a 24 60 6d, /3303/0/5700
121.992141291 127.0.0.1	127.0.0.1	CoAP	96 NON, MID:17277, 2.05 Content, TKN:ff 6a 04 4d 3a 24 60 6d, /3303/0/5700
123.995776262 127.0.0.1	127.0.0.1	CoAP	96 CON, MID:17278, 2.05 Content, TKN:ff 6a 04 4d 3a 24 60 6d, /3303/0/5700
123.998316105 127.0.0.1	127.0.0.1	CoAP	46 ACK, MID:17278, Empty Message
125.992261142 127.0.0.1	127.0.0.1	CoAP	96 NON, MID:17279, 2.05 Content, TKN:ff 6a 04 4d 3a 24 60 6d, /3303/0/5700
125.991978003 127.0.0.1	127.0.0.1	CoAP	96 NON, MID:17280, 2.05 Content, TKN:ff 6a 04 4d 3a 24 60 6d, /3303/0/5700
129.991792102 127.0.0.1	127.0.0.1	CoAP	96 NON, MID:17281, 2.05 Content, TKN:ff 6a 04 4d 3a 24 60 6d, /3303/0/5700
131.992134965 127.0.0.1	127.0.0.1	CoAP	96 NON, MID:17282, 2.05 Content, TKN:ff 6a 04 4d 3a 24 60 6d, /3303/0/5700

FIGURE 14.10 – Emission d'un Observe avec un message CONfirmable

FIGURE 14.11 – Sending an Observe with a CONfirmable message

14.5.2 Lecture d'une instance

En sélectionnant SENML_JSON dans le menu déroulant *Multi Value* et en cliquant sur le bouton *R* à droite de **Instance 0**, différents champs de l'instance se remplissent.

L'échange protocolaire est similaire à précédent. Le serveur LwM2M a émis la requête suivante, où la valeur de la ressource est omis :

```
GET /3303/0
```

et la réponse contient :

```
[{"bn": "/3303/0/", "n": "5601", "v": -6672.6},  
 {"n": "5602", "v": -4573.4},  
 {"n": "5700", "v": -6671.6},  
 {"n": "5701", "vs": "cel"}]
```

On remarque que le nom de base (bn)correspond à l'URI de l'instance et que chaque éléments contient un champs nom (n) contient l'identifiant numérique de la ressource ¹².

14.5.3 Observe

Il est aussi possible de suivre dans le temps l'état d'une ressource, nous allons le faire pour la ressource *Sensor Value* de l'objet numérique *Temperature*. Cliquez sur le bouton *OBS*.

Le serveur LwM2M émet la requête GET comme précédemment, en ayant inséré la l'option **Observe** avec la valeur 0 pour indiquer qu'il souhaite recevoir périodiquement des informations. Les réponses incluent également une option **Observe** dont la valeur est croissante.

Sur Wireshark, vous pourrez vérifier que périodiquement la réponse est envoyée en utilisant un message de type **CONfirmable** plutôt que **NON confirmable** pour vérifier que le client REST sur le serveur LwM2M est toujours actif (cf figure 14.11)

Question 14.5.1: Fin d'observe

Un click sur l'oeil avec la croix permet au serveur d'annuler l'Observe. Quel message est émis ?

12. Le champ (vs) indique que le contenu doit être interprété comme une chaîne de caractères

Question 14.5.2: EXE

Quel message est émis, si on clique sur une ressource de type EXE, comme la remise à zéro des valeurs min et max ?



15. Réponses aux questions

Question 1.8.1 page 25 Les objets communicants sont un tout nouveau domaine, lié aux progrès en miniaturisation des composants électroniques :

- Vrai
- Faux

Les objets communicants ont toujours existé, même avant le déploiement des protocoles de l'Internet. Ce qui est nouveau c'est leur intégration dans les architectures Internet actuelle pour mieux traiter l'information qu'ils produisent.

Question 1.8.2 page 25 Laquelle de ces affirmations est vraie ?

- Il y a très peu de protocoles pour faire communiquer les Objets. Comme l'Internet est une technologie qui s'est très répondu, son succès va permettre aux objets de communiquer.
- Il y a beaucoup de solutions pour permettre à des objets de communiquer, l'Internet des Objets doit permettre de les fédérer.

Chaque métier a créé ses propres protocoles.

Question 1.8.3 page 25 Quelle est la principale source de création des données dans l'IoT ?

- les capteurs
- les nano-ordinateurs (type Raspberry Pi)
- Internet
- les serveurs Web

Les mesures effectuées par les capteurs constituent la principale source de création des données de l'IoT.

Question 1.8.4 page 25 Quels sont les principaux défis technologiques pour l'Internet des Objets (3 réponses) ?

- Avoir une consommation d'énergie faible.**
- Avoir une architecture protocolaire simplifiée.**
- Pouvoir fonctionner sur des systèmes d'exploitation ouverts comme Linux.
- Permettre de sécuriser les données qui peuvent être sensibles.**
- Transmettre en permanence leur état et des valeurs mesurées.

les objets sont généralement contraints en énergie et en capacité. Pour limiter la consommation d'énergie, il ne faut pas émettre ou recevoir tout le temps, donc la transmission en permanence des données ne va pas dans ce sens. De même, un système d'exploitation comme Linux apparaît surdimensionné.

Question 2.1.1 page 29 Dans la pile protocolaire de l'internet, quels protocoles ont pour fonction d'aiguiller les paquets jusqu'à leur destination (2 réponses)

- | | | | |
|-----------------------------------|---|-------------------------------|-------------------------------|
| <input type="checkbox"/> Ethernet | <input checked="" type="checkbox"/> IPv4 | <input type="checkbox"/> MQTT | <input type="checkbox"/> JSON |
| <input type="checkbox"/> IEEE | <input checked="" type="checkbox"/> IPv6 | <input type="checkbox"/> HTTP | |
| <input type="checkbox"/> 802.15.5 | <input type="checkbox"/> UDP | <input type="checkbox"/> CoAP | |
| <input type="checkbox"/> Wi-Fi | <input type="checkbox"/> TCP | <input type="checkbox"/> XML | |

Le protocole IP permet de transporter l'information d'un bout à l'autre du réseau en utilisant les adresses IP contenues dans les paquets. Il existe deux versions de ce protocole IPv4 initialement déployé et IPv6 qui offre beaucoup plus de capacité d'adressage.

Question 2.2.1 page 32 Qu'est ce qui est unique dans le monde (6 réponses) ?

- Un prénom.
- Un nom de famille.
- un numéro de sécurité sociale utilisé en France.**
- un numéro de passeport.**
- un numéro de téléphone portable avec son préfixe international.**
- un numéro complet de compte en banque (IBAN).**
- l'adresse IP de ma machine dans mon réseau privé.
- l'adresse IP d'un serveur Coursera (13.225.34.28).**
- le nom de domaine plido.net.**
- le nom d'une ville.

Ni le prénom, ni le nom de famille, ni leur combinaison ne forment des séquences uniques comme le dirait Jean Dupont. Les numéros de passeport, de sécurité sociale, de téléphone, de compte bancaire sont par construction uniques dans leur espace, mais il pourrait y avoir des recouvrements. C'est pour cela qu'il faut indiquer la source ou l'autorité qui l'a alloué pour garantir l'unicité. Pour le passeport, l'autorité est le pays qui l'a produit. Le numéro de sécurité sociale correspond au numéro d'inscription au Répertoire National d'Identification des Personnes Physiques (RNIPP) (cf. <https://www.service-public.fr/particuliers/vosdroits/F33078>). Le code du pays pour le numéro de téléphone est attribué par l'Union internationale des télécommunications (UIT) puis chaque opérateur a sa zone de numérotation. Pour le compte bancaire, c'est bien sûr la banque ; chaque banque ayant son propre code contenu au début de l'IBAN. L'adresse IP dans un réseau privé n'est pas unique. Le [RFC 1918](#) définit des plages d'adresses que tout le monde peut utiliser localement. Pour sortir sur Internet, il faut un dispositif spécial, appelé NAT, qui va convertir

l'adresse privée en une adresse publique qui, elle, est unique dans le monde. Les serveurs doivent être dans cet espace public et ont donc une adresse unique dans le monde. Les noms de domaines sont uniques dans le monde par construction mais il peuvent être partagés par plusieurs utilisateurs. On peut donc les étendre comme machine1.plido.net, machine2.plido.net... ou, s'il s'agit de la même machine, ajouter un numéro de port après pour indiquer le service : www.plido.net:80, www.plido.net:8080. Quant au nom d'une ville, il n'est pas unique. Il faut aussi préciser le pays, voire la région.

Question 2.2.2 page 34 Le serveur garde un état des précédentes requêtes ?

- Vrai
- Faux

Le serveur répond à une requête puis passe à la suivante. Il ne garde pas d'état. En revanche, une requête peut servir à modifier le contenu d'une ressource, et le résultat de la modification sera gardé.

Question 2.2.3 page 34 Le World Wide Web est basé sur ce principe des états pour :

- fonctionner à la fois sur des ordinateurs et des téléphones portables,
- pouvoir servir un grand nombre de requêtes,
- chiffrer les communications.

Voir le commentaire précédent

Question 2.2.4 page 34 Quels sont les formats qui permettent de représenter des informations structurées (2 réponses) :

- | | | | |
|-----------------------------------|-------------------------------|-------------------------------|-------------------------------|
| <input type="checkbox"/> Ethernet | <input type="checkbox"/> IPv4 | <input type="checkbox"/> MQTT | <input type="checkbox"/> JSON |
| <input type="checkbox"/> IEEE | <input type="checkbox"/> IPv6 | <input type="checkbox"/> HTTP | |
| <input type="checkbox"/> 802.15.5 | <input type="checkbox"/> UDP | <input type="checkbox"/> CoAP | |
| <input type="checkbox"/> Wi-Fi | <input type="checkbox"/> TCP | <input type="checkbox"/> XML | |

Il s'agit de XML et JSON qui permettent d'envoyer des données structurées. Les autres propositions indiquent des protocoles de transport de l'information de niveau 2, 3, 4 et 7.

Question 2.2.5 page 34 Dans l'URI `https://plido.net/unit/definition.html`, quel est le schéma ?

`http` : le Schéma indique comment sera construite l'URI.

Question 2.2.6 page 34 Dans l'URI `https://plido.net:8080/unit/definition.html`, quelle est l'autorité ?

`plido.net:8080` : il s'agit de la séquence globalement unique.

Question 3.3.1 page 40 Dans la première colonne :

- **Le numéro de la trame attribué par Wireshark à sa réception**
- Le numéro de la trame relevé directement dans la trame Ethernet

Ces numéros sont séquentiels, ils sont donc attribué localement par Wireshark. De plus il n'existe aucun champ de cette sorte dans Ethernet.

Question 3.3.2 page 40 Dans la deuxième colonne :

- **L'heure de réception par Wireshark**
- L'instant d'émission de la trame

Comme dans le cas précédent, ce numéro est ajouté par Wireshark, il n'existe pas de champ protocolaire indiquant l'instant d'émission.

Question 3.3.3 page 40 Dans les troisième et quatrième colonnes :

- Les adresses Ethernet des machines.
- Uniquement les adresses IPv4 des machines.
- **Les adresses IPv4 ou IPv6 des machines.**

Wireshark traite de la même manière les adresses IPv4 ou IPv6, elles sont donc affichées dans ces colonnes. L'adresse Ethernet (ou MAC) sur 48 bits n'est pas affichée par défaut dans cet écran.

Question 3.3.4 page 40 Dans la cinquième colonne :

- Le protocole applicatif (niveau 7).
- **Le dernier (de plus haut niveau) protocole reconnu.**
- Le protocole de niveau 4 (ici TCP ou UDP).

Wireshark fournit l'information de plus haut niveau. Dans la figure 3.2 page 39 certaines trames sont indiquées comme transportant le protocole HTTP, tandis que d'autres, généralement les acquittements sont indiqués comme étant de type TCP car elles ne transportent pas de données venant des couches supérieures.

Question 3.3.5 page 40 Dans la sixième colonne :

- La taille en bits de la trame.
- **La taille en octets de la trame.**

L'unité est l'octet.

Question 3.3.6 page 40 Dans la septième colonne :

- **Un résumé des informations transportées par le protocole de plus haut niveau.**
- Les options d'IPv4.
- Le contenu en ASCII du message de plus haut niveau.

Wireshark cherche à interpréter les champs du protocole de plus haut niveau pour offrir un affichage synthétique de l'information.

Question 3.3.7 page 42 Dans la trace suivante, nous avons vu que le serveur répondait aux requêtes du client par un numéro à 3 chiffres. A l'aide du [RFC 7231](#), pouvez-vous attribuer le chiffre

de gauche à une catégorie de notifications :

- | | |
|----------------------------|--|
| 0 <input type="checkbox"/> | <input type="checkbox"/> Redirection |
| 1 <input type="checkbox"/> | <input type="checkbox"/> Erreur coté serveur |
| 2 <input type="checkbox"/> | <input type="checkbox"/> Erreur coté client |
| 3 <input type="checkbox"/> | <input type="checkbox"/> Non attribué |
| 4 <input type="checkbox"/> | <input type="checkbox"/> Succès |
| 5 <input type="checkbox"/> | <input type="checkbox"/> Information |

- 0 : Non attribué
- 1 : Information
- 2 : Succès
- 3 : Redirection
- 4 : Erreur coté client
- 5 : Erreur coté serveur

Question 3.3.8 page 43 Quelle organisation de standardisation a publié ce document ?

- Microsoft
- ISO
- IEEE
- IETF

Le préfixe Request For Comments est caractéristique de l'IETF.

Question 3.3.9 page 44 Est-ce que les en-têtes HTTP ont une taille fixe (vous pouvez aller voir le [RFC 7231](#) qui donne des indications sur le protocole) ?

- L'en-tête tient en une ligne de 80 caractères.
- une ligne blanche sépare l'en-tête du contenu. L'en-tête peut contenir autant de lignes que nécessaire.**

Comme on le voit sur l'exemple donné dans le RFC pour le message de réponse, l'en-tête HTTP contient, une ligne obligatoire, suivie d'options. Leur nombre n'est pas fixé par le standard. Pour les séparer des données, une ligne blanche fait office de séparation.

Question 3.3.10 page 45 Comment sont construites les lignes optionnelles de l'en-tête ?

- mot-clé : valeurs
- un texte non formaté
- mot-clé : longueur des données : valeurs

L'en-tête est de taille variable. Elle comporte une première ligne obligatoire donnant la nature de la requête ou de la réponse, suivie d'informations optionnelles construites sur le format "mot clé : valeur". Une ligne blanche sépare l'en-tête du contenu.

Question 3.3.11 page 45 Dans l'exemple, figure 3.6 page 46, qu'elle est l'adresse Ethernet de la machine émettrice de la trame ?

10:65:30:b0:54:bf. Attention, la trame commence par l'adresse de la destination, suivie de

l'adresse de la source.

Question 3.3.12 page 47 Est-ce que l'adresse Ethernet 14:2e:5e:37:1e:6a que l'on retrouve dans le paquet 3.6 page 46 correspond à l'adresse Ethernet du destinataire du paquet ? A quoi correspond elle ?

Non, l'adresse Ethernet n'est valable que sur ce réseau Ethernet. Pour atteindre le destinataire le paquet devra traverser plusieurs routeurs. Comme la trame a été capturée sur la machine émettrice, cette adresse est donc celle du premier routeur traversé.

Question 3.3.13 page 48 A l'aide de la figure 3.6 page 46 ou de vos captures Wireshark, quels sont les messages impliqués dans la fermeture du connexion ?

La fermeture de déconnexion nécessite l'envoie de quatre message. L'une des extrémité, par force-ment celle qui a ouvert la connexion, envoie un message avec le bit FIN positionné. Il est acquitté par l'autre entité qui a son tour envoie un message avec le bit FIN positionné qui sera acquitté pour définitivement fermer la connexion.

Question 3.4.1 page 49 Quelle URI devez vous entrer dans votre navigateur pour accéder en local à ce serveur.

L'adresse de loopback est 127.0.0.1 et le port est 8080, le chemin d'URI est /. L'URI est donc <http://127.0.0.1:8080/>.

Question 3.4.2 page 49 A l'aide de Wireshark, pouvez vous déterminer dans la réponse les valeurs des options HTTP Content-Type et Server.

Ne pas oublier que le trafic passe par l'interface *loopback*. Pour afficher le trafic sur un port particulier, vous pouvez utiliser le filtre `tcp.port==XXXX`. Don't forget that the traffic goes through the interface *loopback*. To display traffic on a particular port, you can use the `tcp.port==XXXX` filter.

On trouve les valeurs suivantes :

- Server: Werkzeug/2.0.2 Python/3.9.6
- Content-Type: text/html; charset=utf-8. La ressource est codée en HTML en utilisant un codage ASCII sur 8 bits.

Question 4.1.1 page 53 En regardant les échanges de la figure 4.4 page 53 quelle est la valeur mesurée pour l'humidité ?

Seul le premier échange demande la lecture de 2 registres à partir du registre ayant l'adresse 0x0001. Dans la réponse on obtient la valeur des deux registres consécutif (00 DA et 01 D4). Le tableau 4.1 page 53 indique que l'humidité est le second registre, on a donc 01 D4.

Question 4.1.2 page 53 En regardant les échanges de la figure 4.4 page 53 quelle est l'évolution de la température au cours du temps ?

On trouve 3 valeurs mesurées à 19 :11 :48, 20 :02 :39 et 20 :02 :44 : 00 DA, 00 D5 et 00 D5. Soit 21.8°C, 21.3°C et 21.3°C.

Question 4.1.3 page 54 Quel est le taux d'humidité au moment de la mesure ? La documentation précise qu'il s'agit d'un pourcentage avec une précision au dizième de pourcent.

On avait lu la valeur 01 D4 dans le registre 0x02, soit 468 en décimal. D'où un taux d'humidité de 46.8%

Question 4.1.4 page 57 Soit l'échange donné figure 4.9 page 58 correspondant à une requête Modbus et à une réponse. Quel est le numéro de port utilisé par Modbus TCP.

Il y a deux ports dans l'en-tête TCP, mais comme il s'agit d'une requête, il faut prendre le port destination :0x1f6, soit 502 en décimal

Question 4.1.5 page 58 En poursuivant l'analyse du trafic, quelle valeur de registre est demandée.

Le registre 0x36 est demandé, en regardant dans la table 4.3 page 56, il s'agit de la fréquence en Hz.

Question 4.1.6 page 58 En analysant le paquet suivant, comment peut-on vérifier que la réponse peut correspondre à la requête précédente.

Le numéro de séquence 0x000c est le même dans les deux trames.

Question 4.1.7 page 58 Quelle valeur est renvoyée. Est-ce cohérent ?

La valeur du registre est 0x4247e95b correspond à un nombre flottant IEEE 754, en convertissant on obtient la valeur 49.9778862 Hz qui est très proche de la fréquence du réseau électrique européen.

Question 6.8.1 page 81 Quels sont les avantages de CBOR par rapport à JSON (2 réponses) ?

- Il est plus compact dans la représentation des données.**
- Il permet de représenter des nombres flottants.
- Il compresse les chaînes de caractères.
- Il est plus simple à implémenter.**

CBOR ne compresse pas les chaînes de caractères. Il ajoute juste leur longueur. CBOR et JSON codent tous les deux des nombres flottants, ce n'est donc pas un avantage de CBOR. Par contre, le fait d'utiliser des valeurs binaires au lieu de l'ASCII pour représenter les nombres, de ne pas avoir de crochets ou accolades, permet d'avoir une représentation beaucoup plus compacte. Les petites valeurs numériques sont représentées sans surcoût. Réaliser un codeur ou un décodeur de CBOR est beaucoup plus simple qu'en JSON car la représentation des données est beaucoup plus stricte (pas d'espace, pas de retour à la ligne...). Les deux représentations permettent d'utiliser des nombres flottants donc aucune n'a d'avantage sur ce point.

Question 6.8.2 page 81 Un flottant est-il toujours plus compact en CBOR qu'en JSON ? - Vous pouvez vous aider de <https://cbor.me>

- Oui, c'est le but de CBOR.
- Oui, pour les flottants qui ont la partie décimale à 0.
- Oui, pour les flottants de petite précision (jusqu'au centième).
- Oui, pour les flottants de grande précision (6 chiffres après la virgule).**

Un nombre flottant, quelle que soit sa valeur, est représenté par 8 octets en CBOR. En JSON, un nombre flottant est représenté par une chaîne de caractères. Donc "3.0" nécessite 3 caractères, donc plus compacte que CBOR. Mais "3.1415926" est codé sur 9 caractères donc moins compacte que CBOR.

Question 6.8.3 page 81 Soit une chaîne de caractères en CBOR.

- Elle est compressée avec un algorithme entropique (e.g. codage de Huffman).
- **Elle peut contenir des caractères accentués.**
- Chaque caractère est codé sur 6 bits.

A string is a sequence of bytes. It is possible to use representations offering accented characters (even emoticons). However, CBOR does not compress data.

Question 6.8.4 page 82 En CBOR, la taille d'un entier varie en fonction de sa valeur !

- **Vrai**
- Faux
- Ça dépend de la manière dont on a déclaré cet entier.

Oui, un entier inférieur à 23 sera codé sur un seul octet. Pour les valeurs plus grandes, il faut ajouter la longueur.

Question 6.8.5 page 82 En CBOR, un tableau peut contenir des objets de types différents.

- **Vrai**
- Faux
- Ça dépend de la manière dont on a déclaré ce tableau.

Vrai, on a la même flexibilité qu'en JSON en imbriquant n'importe quel type de données dans un tableau.

Question 6.8.6 page 82 On veut définir un tableau de deux éléments comme une fraction. Quel tag devra précéder la structure ? (vous pouvez vous aider du [RFC 8949](#)).

Il s'agit du tag 4, voir le chapitre **3.4.4. Decimal Fractions and Bigfloats** du [RFC 8949](#).

Question 7.1.1 page 87 Pourquoi le programme client ne fonctionne pas ?

- L'adresse IP n'est pas correcte.
- **Il manque un processus de sérialisation de la donnée.**
- La variable t n'est pas définie.
- La variable t ne peut pas être lue.

La variable t pointe vers une représentation en mémoire du nombre flottant. Elle ne peut pas être directement envoyée à un autre équipement. Il faut ajouter une phase de sérialisation qui va permettre de transmettre cette information soit en une chaîne de caractères, soit en une chaîne d'octets. The variable t points to a memory representation of the floating-point number. It cannot be directly sent to another device. It is necessary to add a serialization phase which will make it possible to transmit this information either in a character string, or in a byte string.

Question 7.2.1 page 89 Analysons la séquence reçue : 83fb40341086f3e8b66bfb408f3b7791c8d61ffb403fac

15ba06088e

À quoi correspond l'octet 0x83 qui commence la structure CBOR recue ?

- au codage de l'entier positif 131.
- au codage de l'entier négatif 132.
- à la définition d'un tableau de 3 éléments.
- à la définition d'un map CBOR de 3 éléments.
- à la définition d'un tableau de taille non définie.

0x83 s'écrit en binaire 100-0 0111. 100 est le type majeur pour un tableau. La valeur 00111 est inférieure à 24. Il s'agit donc du nombre d'éléments du tableau, donc un tableau de 3 éléments.

Question 7.2.2 page 89 Dans cette structure, quel est le marqueur CBOR (en hexadécimal) qui indique que l'on a un nombre flottant ?

0xFB, si on l'écrit en binaire on obtient 111-1 1100. Le majeur correspond à la catégorie des flottant et des valeurs spéciales.

Question 7.2.3 page 89 Quelle est la taille de ce flottant en octets ?

8 octets ; la partie mineur 1 1100 indique qu'il s'agit d'un flottant codé sur 8 octets

Question 7.2.4 page 90 Quelle est la taille minimale et maximale de la structure CBOR envoyée, en prenant en compte les valeurs possibles.

- La température peut évoluer raisonnablement entre -30 et +50, soit -3000 et +5000 après la transformation en entier. La taille minimale, si on envoie 0, la taille sera d'un octet. La taille maximale tiennent sur 3 octets (1 pour le type/longueur et 2 pour les valeurs)
- La pression évolue autour de 1000, soit 100 000 après la transformation en entier. La taille sera toujours de 5 octets (1 pour le type/longueur, 4 pour les valeurs)
- Le taux d'humidité évolue entre 0 et 100, soit 0 et 10 000 après la transformation en entier. La taille entre 1 octet et 3 octets.

Si on ajoute le type/longueur 0x83 pour indiquer un tableau de trois éléments, on obtient une taille minimale de $1+1+5+1= 8$ octets et une taille maximale de $1+3+5+3 = 12$ octets.

Question 8.6.1 page 101 A quoi correspond la clé { 'u' : 'Cel' } que l'on retrouve dans la structure précédente ?

Unité = degrés Celcius

Question 8.6.2 page 101 Dans les deux représentations JSON et CBOR, de combien la taille est-elle accrue par l'ajout des mesures effectuées ? d'où viennent ces différences ?

Si l'on regarde le listing précédent, l'ajout des trois mesures fait augmenter la taille de 164 octets pour JSON et 106 pour CBOR. La différence vient de l'utilisation de nombre plus que de chaînes de caractères pour les clés. Ainsi 't' demande 3 caractères en JSON avec les guillemets, codé en CBOR, il faudrait 2 octets, un nombre inférieur à 23 se code sur un seul octet. Il y a également les virgules, espaces et fermeture de crochets qui ne sont pas présent en CBOR. Les nombres flottant comme 19.98 ont une représentation plus compacte en JSON (5 octets) qu'en CBOR où ils consomment 9 octets. Dans tous les cas l'accroissement est fortement dépendant du nom des

éléments. Ici, il faut répéter à chaque fois temperature, humidity et pressure, soit 26 caractères.

Question 8.6.3 page 101 Si on ne s'intéressait qu'à une seule grandeur, par exemple l'humidité. A quoi ressemblerait la structure SenML en JSON ?

Chaque nouvelle entrée ajoute 35 octets à la structure :

```
[{"bn": "device1", "bt": 1640110457.0, "n": "humidity", "u": "%RH", "v": 28.46},  
 {"t": 10.0, "v": 26.86},  
 {"t": 20.0, "v": 26.96},  
 {"t": 30.0, "v": 27.01}]
```

Question 8.6.4 page 103 Pourrait-on utiliser le champ SenML *base value* pour diminuer la taille des données de pression atmosphérique ?

Cela serait possible, si cette ressource était envoyée seule.

Question 9.5.1 page 111 Le module I2C du LoPy dispose d'une fonction scan qui affiche les adresses des secondaires connectés. Comment cette détection est possible ?

Le primaire va tester toutes les adresses possibles et y envoyer une trame, si le bit suivant l'adresse dans la trame n'est pas positionné par le secondaire, il n'y a aucun équipement connecté à cette adresse.

Question 9.5.2 page 111 Est-ce que la norme une adresse qui permet de parler à tous les secondaires en même temps ?

La norme prévoit un *General Call* pour joindre tous les secondaires en utilisant l'adresse 0 (voir chapitre 3.1.12 du standard I2C).

Question 9.6.1 page 115 Que se passe-t-il si dans le programme wifi_temperature.py vous modifiez le pas de mesure ligne 36, pour le mettre par exemple à 60 secondes.

Le programme display_server.py doit être modifié également en ajoutant l'argument period=60 à l'appel de to_bbt.

Question 11.2.1 page 140 Quel message de trace obtenez vous du LNS, si l'Objet n'est pas configuré avec le même AppKey que le LNS ?

```
12:52:11 Join-request to cluster-local Join Server failed MIC mismatch  
12:52:01 Join-request to cluster-local Join Server failed MIC mismatch  
12:51:51 Join-request to cluster-local Join Server failed MIC mismatch
```

Question 11.2.2 page 140 Quel impact sur les batteries ?

Le LoPy va émettre périodiquement tous les 10 secondes un message de type **Join-request**. Le but est pouvoir se connecter, même si la transmission n'est pas optimale. mais en cas de mauvaise

configuration, cela va se traduire par une occupation plus importante du réseau, mais surtout un impact sur l'autonomie de l'Objet.

Question 11.2.3 page 140 Dans les traces précédentes, quelle est la valeur du fPort par défaut utilisé par le LoPy ?

"f_port": 2,

Question 11.2.4 page 140 L'instruction bind permet de modifier le **fPort** pour une transmission. Modifiez le programme pour utiliser le fPort 10 et vérifiez l'effet dans les traces de TTN.

Le programme doit être modifié par exemple après les instructions `setsockopt` en ajoutant la ligne :
`s.bind(10)`

Question 11.5.1 page 150 En vous aidant du lien suivant <https://www.thethingsnetwork.org/docs/lorawan/regional-parameters/> indiquant les paramètres régionaux. Est ce que le programme `lorawan_temperature.py` fonctionnerait sur un réseau LoRaWAN situé en Amérique du Nord ?

Non, la taille maximale des données pour DR0 est de 11 octets.

Question 12.2.1 page 162 Que représente ce message ?

- Une requête GET
- Une requête POST
- Une requête PUT
- Une requête DELETE
- Une notification positive

Le champ code (deuxième octets de l'en-tête CoAP) vaut 0x02, ou 0.02 donc il s'agit d'une requête et d'un POST.

Question 12.2.2 page 162 Quelle est la valeur du champ token ?

- Vide
- 0xb4
- 0xb474
- 0xb47465
- 0xb474656d

Le premier octet 0x40 s'écrit en binaire 0b01_00_0000, soit la version (1), le type (CON) et la taille du champ Token (0), il n'y a donc pas de Token après l'en-tête obligatoire, il y aura directement les options ou le séparateur 0xFF pour indiquer des données.

Question 12.2.3 page 162 Quels éléments de l'URI contient ce message ?

- Aucun
- /temp
- /temp/sens1 et /max

- /temp/sens1/max
- /temp/sens1?max

La séquence des Type/Longueur est 0xb4 Uri-path avec 4 octets de données (temp), 0x05 toujours Uri-path avec 5 octets de données (sens1) et 0x43 donc un type 11+4=15 soit Uri-Query avec 3 octets de données (max). 0xff indique la fin des options.

Question 12.2.4 page 163 Vous avez la ressource suivante :

temperature = 20C

Quelle valeur l'option CoAP Content-type utiliser pour une réponse en CoAP ?

- text/plain
- 0
- 50

L'entier 0 qui correspond à un codage utilisant les caractères ASCII

Question 12.2.5 page 164 Vous voulez recevoir une ressource dans le format SenML ; CBOR. Quelle valeur doit transporter l'option Accept dans la requête ?

112, comme le montre le tableau 12.4 page 163

Question 12.2.6 page 164 Quel code d'erreur retourne le serveur s'il ne peut pas envoyer une réponse dans ce format ? Aidez vous du [RFC 7252](#).

- 4.04 (Not Found)
- 4.02 (Bad Option)
- 4.06 (Not Acceptable)
- 5.01 (Not Implemented))

Cf. Chapitre 5.10.4. du RFC. 4.06 n'est pas facile à trouver. Il faut aller sur le site de l'IANA, aller sur le RFC de CoAP qui pointe sur celui de HTTP pour la définition de ce code qui est rarement utilisé en HTTP. 4.04 n'est pas possible car la ressource existe mais pas au bon format. 4.02 n'est également pas possible, l'option Accept est critique donc doit être connue du serveur. Finalement, il s'agit d'une erreur du client et pas du serveur; donc l'erreur 5.01 n'est pas possible non plus.

Question 13.2.1 page 174 Que se passe-t-il si vous utilisez une requête non confirmable pour demander la ressource /time (mettre l'argument type=CoAP . NON dans la construction de l'en-tête obligatoire).

- Ça plante le serveur.
- Le serveur répond par une requête Confirmable.
- Le serveur retourne un RST car nous n'avons pas programmé ce cas.
- Le serveur répond par une requête Non Confirmable.
- On passe en heure d'hiver.

Question 13.2.2 page 175 Modifiez le programme du serveur pour supprimer le délai de 5 secondes avant une réponse.

Que se passe-t-il quand le client envoie une requête CONFIRMABLE ?

- Le serveur retourne la réponse dans l'acquittement.
- Le serveur retourne une requête NON confirmable.

-
- Le serveur attend quand même quelques secondes pour ne pas saturer le réseau.
 - Le serveur enlève le token de la réponse.

Question 13.2.3 page 175 Modifiez le programme du serveur pour supprimer le délai de 5 secondes avant une réponse.

Que se passe-t-il quand le client envoie une requête NON confirmable ?

- Le serveur retourne la réponse dans l'acquittement.
- **Le serveur retourne une requête NON confirmable.**
- Le serveur attend quand même quelques secondes pour ne pas saturer le réseau.
- Le serveur enlève le token de la réponse.

Question 13.3.1 page 179 Que reçoit-on en réponse à la requête POST ?

- un message ACK
- le statut 2.00 OK.
- **le statut 2.04 CHANGED.**
- rien.

A chaque requête on reçoit une notification REST indiquant que la ressource a été modifiée.

Question 13.3.2 page 179 Modifiez le programme client pour indiquer un content-format JSON. Quelle notification obtenez-vous ?

- un message RST
- le statut 4.04 NOT FOUND.
- **le statut 2.15.**
- le statut 5.00.

Si vous avez 5.00 c'est qu'il y a une erreur dans votre programme coté serveur. La notification pour un format inconnu est 4.15 (Unsupported Content-Format) If you have 5.00 it means that there is an error in your server side program. The notification for an unknown format is 4.15 (Unsupported Content-Format)

Question 14.3.1 page 187 Quelle est la nature de la requête émise par le client :

- GET
- **POST**
- PUT
- DELETE

Le client LwM2M agit comme un client REST et envoie cette requête POST :

Constrained Application Protocol, Confirmable, POST, MID:12142

```
01... .... = Version: 1
..00 .... = Type: Confirmable (0)
.... 1000 = Token Length: 8
Code: POST (2)
Message ID: 12142
```

Question 14.3.2 page 188 Quelle est le chemin de l'URI :

- vide

- /rd
- /lwm2m

Le chemin d'URI est composé d'un seul élément :

Opt Name: 1: Uri-Path: rd

Question 14.3.3 page 188 Quelle est le format du contenu (content-format) :

- text
- XML
- JSON
- **link-format**
- CBOR

Après le chemin d'URI on trouve l'option Content-Format qui contient la valeur 40 soit **link-format** :

Opt Name: 2: Content-Format: application/link-format

Question 14.3.4 page 189 A quelle période voyez-vous d'afficher des messages CoAP ?

La figure 14.3 page 188 montre plusieurs échanges. Le premier à 25.16 secondes est un POST sur /rd. Le second à 78.33 secondes un POST sur /rd/dyUNwRXNOp. Le troisième à 131.37 secondes est identique et les suivants sont identiques. La période d'envoi est de 53 secondes.

Question 14.3.5 page 189 Que signifie b=U ?

- Les données seront émises avec les unités (Units)
- Le référentiel des unités est la représentation Universelle
- **Le protocole sous-jacent est en mode datagramme (UDP)**
- Le client n'est pas référencé (Unreferenced)

Ce paramètre désigne le binding, U indique que CoAP sera transporté sur UDP. Le standard propose également d'autres modes, comme :

- T : TCP. CoAP prévoit également un mode de fonctionnement sur **TCP (RFC 8323)**. Le mode permet par exemple de traverser des réseaux qui restreignent l'usage d' **UDP** pour de soi disant raisons de sécurité ;
- S : SMS. LwM2M est défini par l'Open Mobile Alliance, il est logique d'inclure ce moyen de communication pour gérer un équipement connecté au réseau cellulaire ;
- N : **NON-IP**. Dans ce mode, les messages CoAP sont directement envoyés sur le niveau 2. Il est également connu sous le terme de Non IP Data Delivery (NIDD) dans les réseaux 4G.
- Nous nous basons sur la version 1.1 du standard, la révision 1.2 de LwM2M intègre également de LoRaWAN, MQTT (M), HTTP (H),...

Question 14.3.6 page 189 Que signifie lwm2m=1.1 ? Il s'agit...

- **de la version lwm2m du client**
- de la taille mémoire de l'implémentation (1.1 ko)
- de la version lwm2m du serveur

Il s'agit de la version du client.

Question 14.3.7 page 189 Que signifie lt=300 ?

-
- C'est le nombre maximal d'objets (less than 300)
 - C'est la taille maximale d'un échange (300 octets)
 - **C'est la durée de vie de l'enregistrement d'un objet (lifetime)**

C'est la durée de vie d'une valeur, si elle n'est pas rafraîchie avant ce délai par le client, elle disparaîtra du serveur.

Question 14.3.8 page 190 À quoi sert l'attribut rt ?

- à donner la sémantique (i.e. comment interpréter) de la ressource.
- à indiquer la taille de la ressource en faisant référence à lwm2m.
- à indiquer la version de lwm2m utilisée.
- à aider à déboguer les échanges.

Quand le serveur reçoit le POST du client, il est capable d'associer une représentation au chemin d'URI décrit. Le client et le serveur doivent avoir cette connaissance à priori.

Question 14.3.9 page 194 En vous aidant de la page de description des objets numériques et des ressources¹, que représente l'URI 3301/0/5602 ?

- la température maximale du premier capteur
- l'humidité minimale du capteur 0
- **la luminosité maximale du capteur 0**

En allant sur la page web, on trouve que 3301 correspond à la mesure de la luminosité (*Illuminance*). La description de cet objet numérique est identique à celle de l'objet numérique *temperature*; 3301/3/5602 indique la valeur maximale.

Question 14.3.10 page 194 Que représente l'URI 10340/0/2 ?

- la température en Fahrenheit de l'objet 10340
- la longitude dans des coordonnées GPS
- **le statut actif ou non d'une caméra**

En allant sur la page web, on trouve que 10340 correspond à un objet numérique *Camera* défini par la société CloudMinds. La description de cet objet numérique montre que la ressource 2 correspond au status (1 : *Enabled*, 0 : *Disabled*.)

Question 14.3.11 page 194 Quelle URI permet d'accéder à l'élément contrôlant la variation lumineuse (*Dimmer*) d'un éclairage (*Light Control*) ? Quelle valeur maximale peut prendre cette ressource ?

3311/0/5851 La valeur maximale est de 100.

Question 14.3.12 page 194 Quel serait le chemin d'URI pour obtenir la fréquence d'un courant électrique sur la deuxième phase ?

3318/1/5700

1. <http://www.openmobilealliance.org/wp/OMNA/LwM2M/LwM2MRegistry.html#resources>

Question 14.3.13 page 194 Dans le premier échange capturé :

```
</>;ct="60 110 112 11542 11543";rt="oma.lwm2m",
</1>;ver=1.1,
</1/0>,
</3>;ver=1.2,
</3/0>,
</6/0>,
</3303>;ver=1.1,
</3303/0>,
</3442/0>
```

1 : Device

3 : LwM2M Server

6 : Location

3303 : Temperature

3442 : LwM2M v1.1 Test Object

Question 14.4.1 page 195 Que répond le serveur (cf. figure 14.6 page 196) ?

- Il acquitte juste le message.
- Rien. Nothing.
- **Il renvoie une URI qui servira par la suite à identifier l'objet. It returns a URI that will be used later to identify the object.**
- Il retourne les données de chiffrement des messages. It returns the message encryption data.

La réponse contient l'option **Location-Path** ce qui permet de préciser où la ressource fournie par la client a été référencée sur le serveur. Comme pour **URI-Path**, il s'agit d'une option qui peut se répéter. Dans l'exemple, figure 14.6 page 196, l'URI fournie est donc /rd/dyUNwRXNOp.

Cette URI servira par la suite d'identifiant interne pour le client.

Question 14.4.2 page 195 Si on laisse la plate-forme fonctionner sans intervenir, que voit-on sur l'analyseur réseau ?

- Rien.
- Le capteur envoie des informations indiquant un changement de valeur mesurée.
- Le client envoie les valeurs mesurées même s'il elles n'ont pas changé.
- **Le client envoie un message vide vers le serveur pour indiquer qu'il est toujours présent.**
- Le serveur envoie un message vide vers ses clients pour savoir s'ils sont toujours présents.

Toutes les 53 secondes, le client envoie un POST vers la ressource créée à l'enregistrement. Ce POST est vide. Il sert à indiquer au serveur que le serveur est toujours présent. Le serveur acquitte ce message indiquant au client qu'il est aussi accessible.

Question 14.5.1 page 199 Un click sur l'oeil avec la croix permet au serveur d'annuler l'Observe. Quel message est émis ?

Quand le serveur LwM2M reçoit un Observe pour une session qui a été annulée, il répond avec un

message de type *ReSeT*, la valeur du champ *Message ID* permet au client LwM2M de savoir quelle session d'Observe annuler.

Question 14.5.2 page 199 Quel message est émis, si on clique sur une ressource de type EXE, comme la remise à zéro des valeurs min et max ?

Cela ne change rien au protocole, un POST sur l'URI de la ressource est émis par le serveur LwM2M.

Index

Symbols

<i>broker</i>	141
3GPP	27, 60
4G	60
5G	60
6LoWPAN	62, 63

A

ABP	136
Accept	161, 163, 198
ACK	152, 154, 172, 198
ADSL	17
AF_INET	123
AF_LORA	137
AF_SIGFOX	123
aiocoap	167
analyseur de spectre	118
aoicoap	183
API REST	94
AppEUI	134
appEUI	132
AppKey	132, 134, 139, 140, 210
Arduino	21
AS	133

B

backend	117, 132
balise	69
base64	67, 139
Beebotte	94
binascii	67
BLE	61
Bluetooth	61
BME280	104, 114
bn	198, 199
boot.py	108
Bouygues Télécom	131

C

callback	126
CBOR	63, 73, 74, 151, 163, 177
cbor2	74, 75
chirpstack	132
CLC	112
clé	71
CoAP	63, 130
Code	153

coil 51
collision 118
compression 62
compteur électrique 55
CON 152, 154, 156, 166, 172, 199
Content-Format 161, 214
Content-format 163, 177, 178, 181
Content-Type 49, 206
Cosem 63
couche 26
couche d'adaptation 62
CRC 51, 53
CSV 65, 151
ct 190

D

Data Rate 149
DCC 112
DELETE 34, 153
devAddr 136, 139
devEUI 132, 139
discrete input 51
DLMS 63
DR 137
Duty Cycle 60, 137

E

Empty 171
Epoch 96, 122
ETag 161
Ethernet 39, 45
except 137

F

Flask 48, 128, 143
forward_data 143, 145
fPort 140, 147, 211
fragmentation 62
Fréquence 56
FTP 108

G

GET 33, 126, 153
GND 112

H

HEAD 33
hexadécimal 37
holding register 51
Horizontale 17
HTML 24, 30, 69
HTTP 22, 28, 29, 33, 39, 40, 43, 63, 151, 152
HTTPS 33, 63

I

I2C 112, 115
IANA 153, 189
IBAN 33, 202
IEEE 27
IEEE 754 55, 81
IEEE 802.15.4 61, 63
IEEE 802.3 45
IETF 21, 24, 25, 63
If-Match 161
If-None-Match 161
ifconfig 109, 141
img 69
input register 51
Intensité 56
IoT 16, 18
IP 27, 28, 32, 60, 62, 202
IPv4 28, 202
IPv6 28, 62, 63, 202
IRI 30
ISBN 32
ISO 26, 37, 45, 59

J

java 185

Javascriptt 70
 Join 136
 Join-accept 139
 Join-request 139, 210
 JSON 29, 30, 63, 70, 71, 73, 74, 151, 163
 JSON-LD 73

K

kpn_senml 115

L

LCIM 23
 Least Significant Bit 182
 Leshan 185
 link-format 189, 214
 Linky 63
 Linux 21, 105
 LNS 22, 60, 131
 Location-Path 161, 216
 Location-Query 161
 loopback 37, 84, 109, 127, 143, 145, 186
 LoPy4 104
 LoRa 131
 LoRaWAN 60, 105, 131
 LPWAN 60–62, 116
 LwM2M 163, 185, 190

M

Mac OS 105
 maillé 59
 Matching_sent 182
 Max-Age 161
 Mesh 59
 Message ID 153
 micro-python 104
 Modbus 50, 110, 192
 Module Python
 aiocoap
 Resource, 169
 TimeResource, 175

aoicoap
 Message, 169
 beebotte
 BBT, 95
 writeBulk, 97
 binascii
 hexlify, 67, 69, 144
 unhexify, 122
 unhexify, 67
 unhexlify, 126
 unhexlify, 136
 BME280
 read_raw_temp, 113
 read_temperature, 113
 cbor2
 dumps, 75, 88, 178
 loads, 75, 81, 126, 179
 CoAP
 add_option, 171
 add_payload, 176
 new_header, 171
 send_ack, 176
 datetime
 now, 96
 timetuple, 96
 Flask
 run, 48
 json
 dumps, 73, 79, 88
 loads, 73, 88, 121
 kpn_senml
 dumps, 115
 SenmlPack, 100
 SenmlRecord, 100
 lora
 has_joined, 137
 join, 136, 137
 machine
 scan, 111, 210
 network
 LoRa, 132
 lora.mac(), 67
 Sigfox, 117, 123
 pprint
 pprint, 72
 pycom

rgbled	136
requests	
HTTPBasicAuth	121
post	147
select	
select	144
socket	
bind	85, 140, 211
recv	138
recvfrom	145
recvfrom	85, 144
send	137
sendto	87, 145
setsockopt	137, 211
socket	85, 117
str	
decode	69
encode	87
format	88
time	
maketime	96
MQTT	22, 60, 141
N	
n	199
NAT	126, 141, 202
net-tools	109
NGW	60
NIDD	214
No response	180
No-Response	161
No-response	181
no_response	184
NON	152, 154–156, 166, 199
NON-IP	214
not_sent	182
NXP	110
O	
objet	71
Observe	161, 164, 198, 199
OMA	163, 185, 190, 192
option	158
Orange	131
OTAA	136
P	
PAC	116, 117
Packet Forwarder	148
PATCH	34
pip	48
port	28
POST	33, 126, 153
Programmes micro-python	
BME280.py	113
CoAP	170
coap_empty_msg.py	170
coap_full_sensor.py	182, 183
coap_get_time.py	171
coap_get_time2.py	172, 173
coap_get_time3.py	173, 174
coap_get_time4.py	174
coap_port_temp1.py	176
coap_post_temp1.py	176
coap_post_temp2.py	178
coap_post_temp4.py	180
config.json	148
config_sigfox.py	120
lorawan_devEUI.py	132, 134, 136
lorawan_send_and_receive.py	134, 136, 138
lorawan_temperature.py	149, 150, 211
main.py	148
sending_client.py	110
sigfox_id.py	116, 117
sigfox_temperature.py	122, 123
wifi_conf.py	108, 148
wifi_temperature.py	114, 115, 122, 210
Programmes Python	
cbor-array.py	78
cbor-integer-ex1.py	75
cbor-integer-ex2.py	76
cbor-mapped.py	79
cbor-string.py	77
cbor-tag.py	81
coap_basic_server1.py	167, 169
coap_basic_server2.py	175, 176

coap_basic_server3.py	178	ReSeT	166
coap_post_temp4.py	180	REST	29, 33, 63, 69, 151
coap_server.py	184	RESTfull	33
config_bbt.py	94	RFC	24
device_message.py	124	RFC 1918	202
device_messages.py	120, 121, 123	RFC 2396	160
display_receive_and_send.py	145, 150	RFC 3986	31
display_server.py	94, 95, 98, 115, 127, 130,	RFC 4944	62
150, 210		RFC 5988	190
display_sigfox.py	124, 126	RFC 6282	62
example_json.py	72	RFC 6690	189, 190
generic_coap_relay.py	183	RFC 7228	21
generic_relay.py ..	128–130, 141, 144–146,	RFC 7230	43
148, 167		RFC 7231	43, 45, 204, 205
minimal_client1.py	87	RFC 7252	63, 155, 156, 164, 190, 212
minimal_client2.py	87	RFC 7641	164
minimal_client3.py	88	RFC 791	47
minimal_client4.py	88	RFC 8259	70, 71
minimal_client5.py	89	RFC 8323	214
minimal_client6.py	90	RFC 8376	60
minimal_humidity1.py	91, 92	RFC 8428	82, 83, 99
minimal_humidity2.py	92, 93	RFC 8724	62, 181
minimal_senml_client.py	100	RFC 8949	73, 82, 208
minimal_senml_server.py	102	RNIPP	202
minimal_server.py	84, 88, 109	routeur	27
simple_server.py	48	RS-485	50, 52
ttn_config.py	145, 146	RSSI	139
virtual_sensor.py	86	RST	152, 171
Proxy-Scheme	161	rt	190
Proxy-Uri	161	RTT	156
publish/subscribe	34		
Puissance	56		
PUT	33, 153		
Pycom	104	S	
Pygate	147	sablier	27
pymakr	105–107	SCEF	22, 60
		SCHC	62
		SCL	112
		SDA	112
QModMaster	52	SenML	82, 99, 163, 198
		Sigfox	60, 105, 116, 131, 140, 181
		silos	17
		Size1	161
		span	69
Raspberry Pi	21	spotify	32
		SSID	108

Q**R**

- star 59
STIC 18
Storage integration 141
str 87
sérialisation 65
verticale 17
VIN 112
virtual_sensor 86
Voltage 56
VPS 126, 127
vs 199

T

- tableau 71
tag 79
TCP 28, 29, 47, 63, 214
telnet 108
TKL 152, 158
TLV 158, 163, 198
TNT 17
to_btt 126
Token 156, 166, 198
topic 141
try 137
TTN 131
Type 152

W

- W3C 24, 25, 69, 73
Webhooks 141
Wi-Fi 45, 61, 108
Windows 105
Wireshark 37, 55, 56, 127, 185
WWW 16

X

- XML 29, 63, 69, 70, 73, 192
XY-MD02 51

U

- UDP 28, 47, 63, 214
UIT 202
UNB 118
URI 29–33, 60, 63, 69
Uri-Host 161
URI-Path 216
Uri-Path 161, 172, 214
Uri-path 162
Uri-Port 161
Uri-Query 161
Uri-query 162
URL 32, 47
URN 32
USB 52, 105

- étoile 60, 62

V

- v 198
ver 190



Bibliographie

- [Fie00] Roy Thomas FIELDING. “REST : Architectural Styles and the Design of Network-based Software Architectures”. Doctoral dissertation. University of California, Irvine, 2000.
URL : <http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm> (cf. page 29).
- [TM03] Andreas TOLK et James A MUGUIRA. “The levels of conceptual interoperability model”. In : *Proceedings of the 2003 fall simulation interoperability workshop*. Tome 7. Citeseer. 2003, pages 1-11 (cf. page 23).