RAPPORT DE PROJET DE FIN D'ETUDES

LEFEVRE François

Déploiement d'un réseau LoRaWAN et analyse de vulnérabilités

Encadrants

Thomas Vantroys *Xavier Redon* *Alexandre Boé*

Année scolaire 2017-2018 IMA5

Sommaire

Remerciements

Je tiens à remercier Thomas VANTROYS, ainsi que Xavier REDON et Alexandre BOE pour m'avoir encadré tout au long de ce travail et apporté les conseils dont j'avais besoin.

Je remercie tout particulièrement Thierry FLAMEN qui a été d'une grande aide pour la conception de cartes électroniques.

Je finirai par remercier toutes les personnes ayant réalisé différents travaux sur ce sujet et qui ont été source d'inspiration pour moi, en particulier M. Brocaar pour la conception de son implémentation LoRaWAN et son attention lorsque j'avais des questions sur celle-ci.

Introduction

Pour clôturer cette dernière année et notre formation à Polytech Lille en spécialité Informatique Microélectronique et Automatique (IMA), nous avions pour mission de réaliser un projet de fin d'études sur l'un des sujets proposés par nos professeurs. Pour ma part, j'ai choisi celui imposant le déploiement d'un réseau LoRaWAN pour pouvoir par la suite évaluer les vulnérabilités sur ce réseau et les attaques possibles. C'est un sujet orienté sécurité informatique, un domaine que je souhaitais vraiment approfondir, ce qui était la principale raison de ce choix.

Le monde de l'internet des objets devient de plus en plus important et les prévisions montrent que le nombre d'objets connectés à travers le monde sera décuplé d'ici 2020, passant de plusieurs millions à plusieurs milliards. La problématique concernant la sécurité des réseaux LPWAN (Low Power Wide Area Network) devient donc incontournable. Ce type de réseau que l'on retrouve essentiellement dans ce cas de figure regroupe différents protocoles de communication, dont LoRaWAN. Des vulnérabilités sur cette spécification pourraient amener des enjeux liés à la confidentialité, l'intégrité et la disponibilité des données d'utilisateurs. C'est sur cet aspect que tout l'intérêt du projet se situe.

Pour résumer toutes les tâches accomplies lors de ce travail, nous commencerons par donner le cahier des charges et présenter généralement le protocole LoRaWAN. Ensuite, nous évoquerons tout ce qui a pu être effectué au cours du projet, de l'implantation du réseau jusqu'aux tests d'attaque sur ce même réseau. Enfin, nous conclurons en réalisant un bilan de ce qui a été réalisé et en donnant des évolutions possibles sur le sujet.

Cahier des charges

Description

Comme énoncé précédemment, l'objectif principal du projet est de déployer un réseau LoRaWAN pour pouvoir ensuite évaluer les attaques possibles sur le réseau. Au sein de cette architecture on retrouve une station de base, formée par un concentrateur ou récepteur LoRa associé à une Raspberry Pi, ainsi que divers capteurs embarquant une radio LoRa (émetteur) pour pouvoir échanger avec la passerelle ou station de base, qui seront au nombre de trois.

La passerelle doit être capable de récupérer tous les paquets LoRaWAN émis par les nœuds ou capteurs et de les retransmettre à un serveur qui lui est rattaché afin de pouvoir effectuer un traitement sur ces paquets et d'en tirer de l'information. On notera bien évidemment que tout ce réseau doit être implémenté en respectant la certification LoRaWAN, que nous présenterons par la suite. De plus, sur demande des encadrants, l'implémentation que l'on choisira ne devra pas dépendre d'un réseau public comme The Things Network (TTN).

Une fois que ce réseau est implanté, une évaluation des vulnérabilités du réseau sera opérée. On se concentrera plus précisément sur les faiblesses du protocole LoRaWAN en lui-même, en laissant de côté les attaques plus générales comme prendre le contrôle d'une machine. Nous nous intéresserons principalement aux attaques par rejeu. Pour réaliser des attaques, nous utiliserons une seconde passerelle identique à celle de notre réseau pour pouvoir capturer des paquets en continu. De plus, nous retirerons un nœud du réseau pour pouvoir émettre les paquets que nous souhaitons.

Matériel mis à disposition

- Deux Raspberry Pi 3 avec deux concentrateurs LoRa ic880a-SPI : un ensemble sera utilisé pour l'implantation du réseau et le second pour la réalisation d'attaques.

Figure 1 : Raspberry Pi 3 + concentrateur LoRa

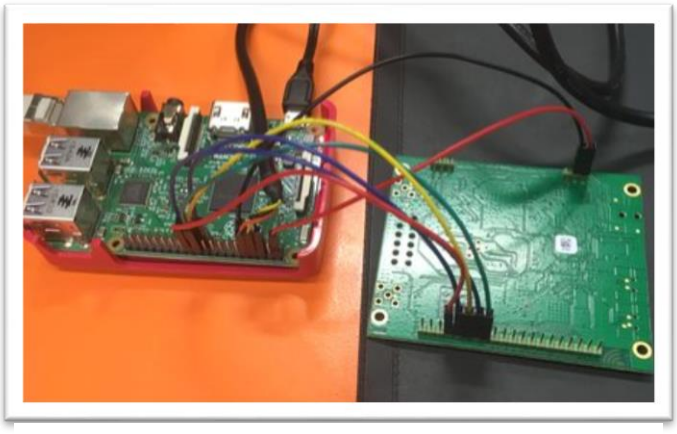

Figure 2 : Montage du concentrateur sur la Raspberry

- Trois microcontrôleurs Nucleo STM32 avec trois shields LoRa (deux sont des I-NUCLEO-LRWAN1 et le dernier est un SX1276MB1MAS) : pour la phase de tests d'attaque, on utilisera un de ces trois modules.

Figure 3 : STM32 + shield SX1276MB1MAS

Figure 4 : STM32 + shield I-NUCLEO-LRWAN1

Etat de l'art sur le protocole LoRaWAN

Après avoir correctement défini notre cahier des charges ce qui nous permettra d'avoir un fil conducteur pour la suite du projet, nous pouvons passer à la présentation de LoRaWAN.

Cette étape est primordiale et indispensable si l'on souhaite pouvoir déployer un réseau respectant ce protocole dans de bonnes conditions, pour savoir si l'on suit bel et bien les règles énoncées et comprendre concrètement ce qui se passe au sein de l'architecture. De plus, pour réaliser des attaques et des analyses sur la vulnérabilité d'un protocole de communication, il est fortement recommandé de connaître la structure des trames échangées, la manière dont celles-ci sont transmises. Chaque information sur la certification peut toujours se transformer en faiblesse. Il faut donc être au point sur LoRaWAN.

Présentation générale

Mis en place par la LoRa Alliance, LoRaWAN est un protocole de communication dédié à l'internet des objets puisque ses principales caractéristiques sont d'être peu coûteux à mettre en place et d'avoir une faible consommation énergétique. Son principal avantage est de permettre à des équipements de communiquer sur de longues distances, à savoir plusieurs kilomètres, et que ceux-ci aient une autonomie conséquente. Le débit des communications n'est pas l'intérêt principal du protocole comme plus généralement pour l'internet des objets. Celui peut varier de 0,3 à 50 kbps.

D'un point de vue électronique, la certification s'appuie sur une technique de modulation par étalement de spectre appelée LoRa. Grâce à cet aspect, des communications ayant des débits différents ne peuvent pas interférer entre elles, ce qui permet à une station de base de gérer un millier d'équipements en théorie. L'une des forces de LoRaWAN provient de ce paramètre et surtout du fait que cela est géré automatiquement par le serveur réseau par un procédé de « débit adaptatif » (ADR). On optimise la durabilité de chaque équipement ainsi que la capacité du réseau en choisissant le débit idéal.

Au niveau de l'architecture du réseau, on retrouve une topologie en étoile où des « nœuds », qui sont les divers équipements souhaitant communiquer au sein du réseau, vont transmettre de l'information à des serveurs applicatifs ou « application servers » en passant par des passerelles ou « gateways ».

Les nœuds utilisent la modulation LoRa pour émettre des paquets par voie RF vers les gateways. Ces passerelles sont forcément rattachées à un serveur réseau ou « network server », qui s'occupera du traitement des paquets et se chargera de les retransmettre ou non vers le(s) serveur(s) applicatif(s). La communication entre passerelles et serveurs est établie via le protocole IP, en utilisant donc un réseau Ethernet ou 3G par exemple. Les informations, une fois transmises au serveur applicatif également via IP, sont à la disposition des utilisateurs et peuvent être exploitées comme on le souhaite. On notera qu'il arrive que le « network server » et le « application server » soient sur la même machine et ne communiquent donc qu'en local. Ce sera le cas pour notre réseau.

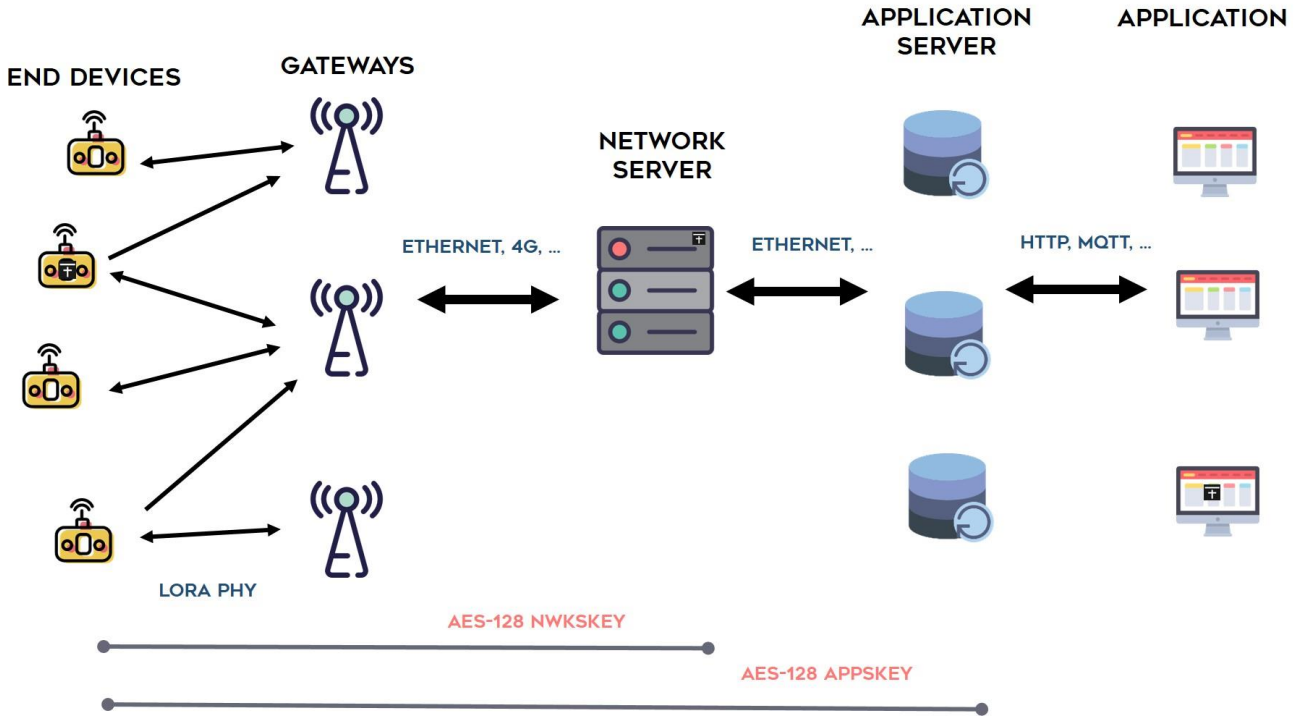

Figure 5 : Architecture d'un réseau LoRaWAN

Différentes classes pour différentes applications

LoRaWAN assure alors une communication bidirectionnelle et définit trois classes d'équipements pour cela : classe A, B ou C. La différence entre ces classes réside principalement dans le nombre de fenêtres utilisées par un nœud pour la réception des messages envoyés par le réseau. Le choix d'une classe dépendra de l'application finale pour un équipement.

Un équipement de classe A est celui qui possédera la consommation la plus faible. Le seul moment où il pourra recevoir des messages provenant du serveur sera juste après une émission. Il ouvrira alors des fenêtres de réception d'un temps précis pour recevoir une réponse du serveur, comme une confirmation pour un message envoyé par exemple (dit aussi « acknowledgment »). Une application de classe A pourrait être un capteur de température envoyant des données de manière périodique.

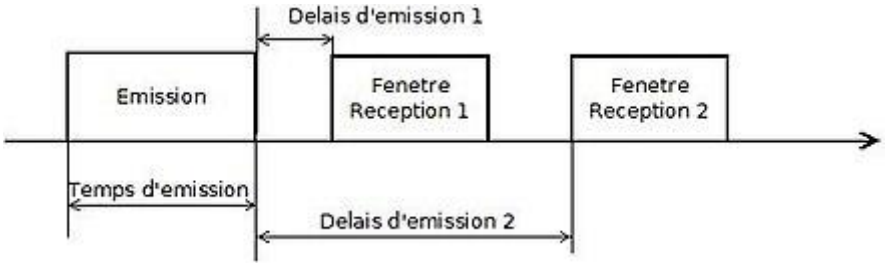

Figure 6 : Fenêtres de réception après émission pour la classe A

Pour les équipements de classe B, on assure un compromis entre consommation énergétique et le besoin de recevoir des trames sur des intervalles précis. Le principe est le même que pour un équipement de classe A mise à part que les fenêtres de réception ne seront ouvertes qu'à des intervalles programmés. Pour cela, c'est la passerelle qui enverra des balises ou « beacons » pour indiquer aux nœuds quand ils doivent ouvrir des fenêtres de réception. Un exemple pour cette classe serait un actionneur qui doit être contrôlé à distance.

Pour terminer avec la classe C, c'est celle qui consomme le plus mais qui permet une communication bi-directionnelle constante. En effet, les équipements de cette catégorie possèdent des fenêtres d'écoute permanentes. De manière générale, ce sont les gateways qui seront de classe C pour pouvoir récolter les messages de tous les équipements en continu.

Sécurité du protocole

En plus d'assurer une bonne gestion de tous les équipements d'un réseau, LoRaWAN garantit également la sécurité du réseau ainsi que la confidentialité des données, un aspect essentiel en ce qui concerne l'internet des objets.

Pour cela, le protocole a recours à deux chiffrements AES-128 en utilisant deux clés différentes. La première, qui est la clé de session réseau ou « Network Session Key » (NwkSkey), assure l'authenticité d'un nœud sur le réseau par un premier chiffrement tandis que la seconde, clé de session applicative ou « Application Session Key » (AppSKey), garantit la confidentialité des données transmises sur le réseau par un second chiffrement.

Les données utiles ou « payload » que l'équipement souhaite transmettre sont donc d'abord chiffrées avec l'AppSKey pour éviter qu'elles ne soient transmises en clair. Il faudra ensuite ajouter toutes les en-têtes nécessaires pour avoir une trame LoRaWAN correspondant à la spécification. On chiffre ensuite cette trame finale avec la NwkSKey pour calculer un « Message Integrity Code » (MIC) que l'on concaténera à la fin du paquet précédemment créé. Cela permettra au serveur réseau de vérifier si la trame provient réellement d'un nœud du réseau, en calculant également de son côté le MIC du paquet reçu et en le comparant avec celui-ci rattaché au paquet.

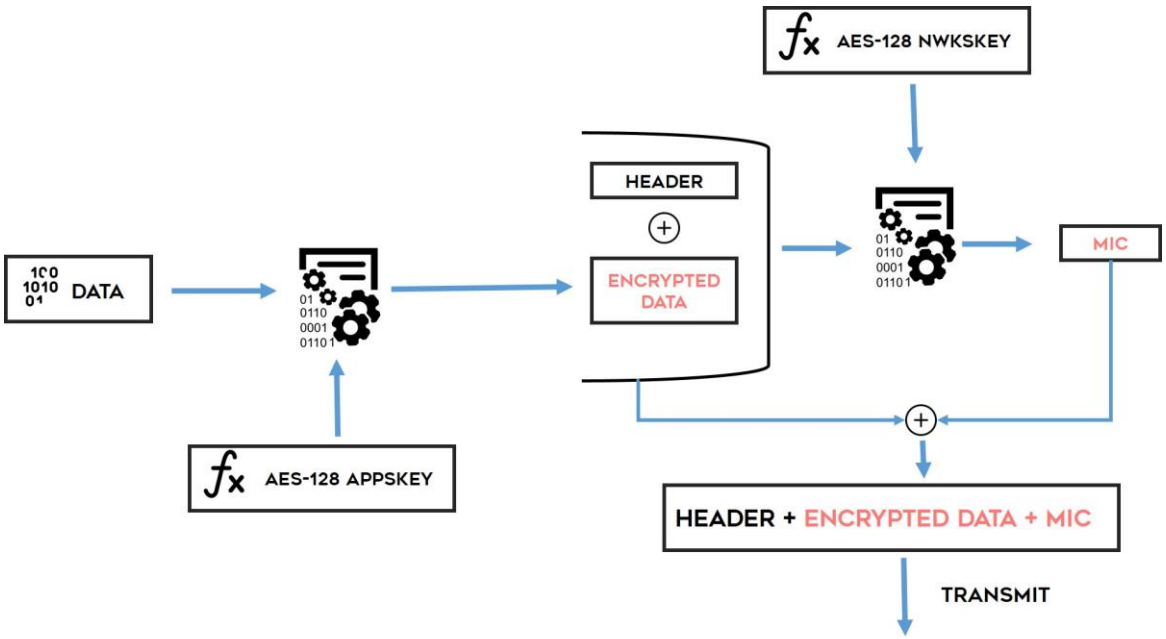

Figure 7 : Chiffrement d'un message LoRaWAN

D'un côté, la clé de session applicative n'est connu que du fournisseur de l'application, ce qui empêche un tiers voire même l'opérateur d'avoir accès aux données des utilisateurs. D'autre part, la clé de session réseau est partagée avec l'opérateur réseau ainsi que les fournisseurs d'applications autorisés.

En plus de sécuriser le réseau et les communications réalisées par les équipements, ces clés servent également à activer ces équipements sur le réseau.

Activation des équipements

Over-The-Air Activation (OTAA)

Cette méthode est la plus sécuritaire des activations puisqu'elle permet dériver les clés de session à partir d'un secret partagé par l'équipement et le serveur applicatif, que l'on appelle AppKey ou clé applicative. Cette technique s'apparente légèrement à un mode connecté tel que TCP, puisque l'équipement va d'abord transmettre une requête pour rejoindre le réseau (« join request »). Si l'équipement est autorisé par le serveur, il reçoit alors une confirmation pour sa requête (« join accept »).

Une fois ces deux étapes effectuées, l'équipement et le serveur auront les informations nécessaires pour calculer les deux clés de session chacun de leur côté et communiquer avec celles-ci par la suite de manière sécurisée. L'intérêt de cette méthode est que les clés de session seront de nouveau générées si l'on initialise une nouvelle session, c'est-à-dire l'envoi d'un nouveau « join request ».

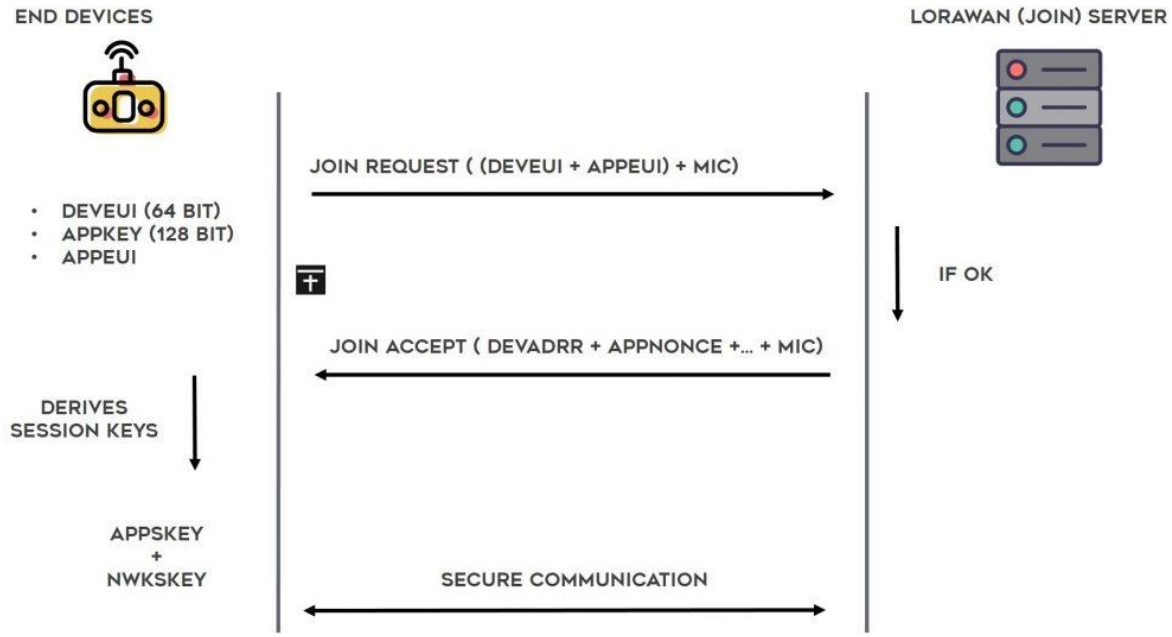

Figure 8 : Activation par la méthode OTAA

Activation By Personalization (ABP)

Cette méthode est plus simple à mettre en place que la précédente et est plutôt réservée à des phases de tests. Cependant, elle n'assure pas une sécurité aussi complète que la méthode OTAA. En effet, on ne passe par la procédure de « join », un nœud peut directement envoyer des messages au serveur. Par contre, cela implique qu'il n'y a pas de génération de clés de session. Celles-ci sont prédéfinies à l'avance sur le serveur applicatif et l'équipement en question. Dans ce cas de figure, une session possédera toujours les mêmes clés, ce qui nuit grandement à la sécurité de l'équipement. Si jamais les clés sont compromises, la sécurité de toutes les prochaines sessions sera compromise tant qu'on ne changera pas la valeur de ces clés.

Format des trames

Comme tout autre protocole, les trames LoRaWAN respectent un format bien précis et possèdent des champs particuliers. Dans les en-têtes, nous retiendrons principalement le champ Fcnt ou « frame counter » qui s'incrémente à chaque message envoyé. C'est en utilisant essentiellement une faille liée à cette caractéristique que nous pourrons réaliser des attaques lors du projet.

La structure des trames est définie comme ceci selon la dernière spécification LoRaWAN (v1.1.0) :

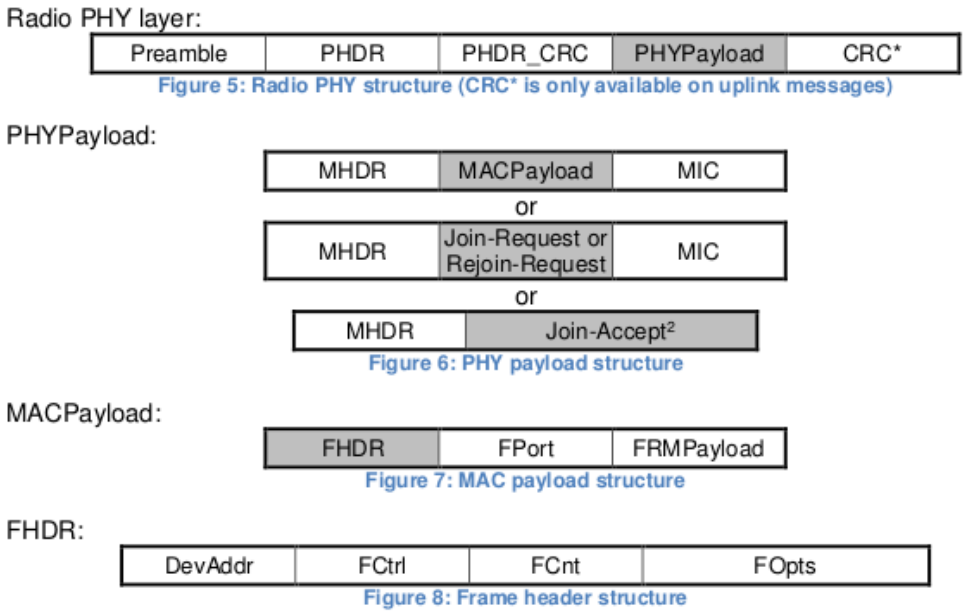

Figure 9 : Structure détaillée des trames LoRaWAN

Réalisation du projet

Maintenant que le protocole LoRaWAN a été clairement présenté, nous allons pouvoir passer à toutes les tâches effectuées au cours de ce projet de fin d'études.

Conception du shield Raspberry-concentrateur

Pour débuter ce projet, nous avions tout d'abord pour mission de trouver sur le net une carte électronique permettant de relier notre concentrateur LoRa à notre Raspberry pour éviter de constamment utiliser des fils, comme nous l'avons fait pour présenter le matériel au début du rapport.

Nous avons alors trouvé quelques fabricants proposant cette solution. Il ne nous reste donc plus qu'à commander ce produit pour passer à l'implantation de notre réseau. Cependant, nous nous sommes vite rendu compte que le produit était en rupture de stock sur tous les sites trouvés. La solution idéale était donc de créer cette carte nous-mêmes, ce qui réduira également le coût du matériel, étant donné que nous pouvons les fabriquer à l'école.

Cette étape de conception s'est donc ajoutée à notre cahier des charges, ajoutant une partie d'électronique à notre travail. Bien que le design de cette carte fût largement réalisable, nous avons préféré récupérer les fichiers de design sur le site d'un fabricant pour aller au plus vite sur le cœur du projet et ne pas perdre de temps sur ce point.

Ces fichiers ont ensuite été transmis au service EEI de Polytech Lille pour que la carte puisse être gravée. Par la suite, nous avons passé un certain temps à souder les connecteurs permettant de relier les broches de la Raspberry Pi à celle du concentrateur. La difficulté concernant cette étape fût qu'il était nécessaire de souder des fils métalliques directement sur les pistes pour conduire le courant d'une couche à l'autre. Effectivement, la carte réalisée est une carte multicouches, ce qui signifie que nous avons des pistes démarrant de la couche supérieure allant jusqu'à la couche inférieure ou inversement. Etant donné que nous n'avons pas de carte métallisée au service EEI qui gère cette conduction du courant, il était alors nécessaire de réaliser ces soudures supplémentaires.

Nous avons alors en notre possession un shield qui est fonctionnel et qui nous permet d'avoir une passerelle LoRa beaucoup plus compact, ce qui peut s'avérer utile s'il est nécessaire d'implanter cette passerelle dans un cas réel et de la placer dans une boîte par exemple.

Une fois cette étape validée, nous avons pu passer au déploiement de notre réseau LoRaWAN.

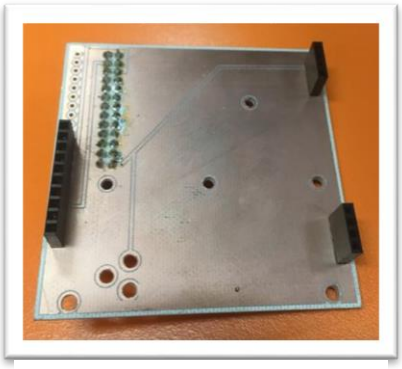

Figure 10 : Couche supérieure du shield

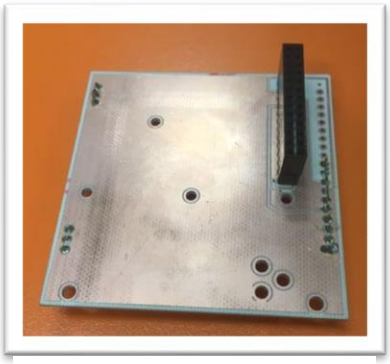

Figure 11 : Couche inférieure du shield

Figure 12 : Montage final pour notre station de base

Premiers pas et test du matériel

Avant d'entrer directement dans le vif du sujet et de mettre en place notre réseau basé sur LoRaWAN, nous avons d'abord appris à connaître le matériel en réalisant des tests simples pour vérifier que chaque module puisse communiquer normalement.

Test des nœuds

Nous avons commencé cette phase de test par vérifier que notre nœud SX1276 était capable d'envoyer des trames simples en récupérant un programme de ping-pong entre deux nœuds de ce type. Nous avons donc utilisé deux nœuds SX1276 pour cela même si on n'en retrouvera qu'un seul sur l'architecture finale. Ce programme peut être retrouvé en suivant le lien donné dans la bibliographie. On notera que tous les programmes tournant sur les nœuds sont compilés en utilisant le compilateur en ligne de Mbed, ce qui nous permet d'obtenir le binaire à copier directement sur le microcontrôleur comme sur une clé USB.

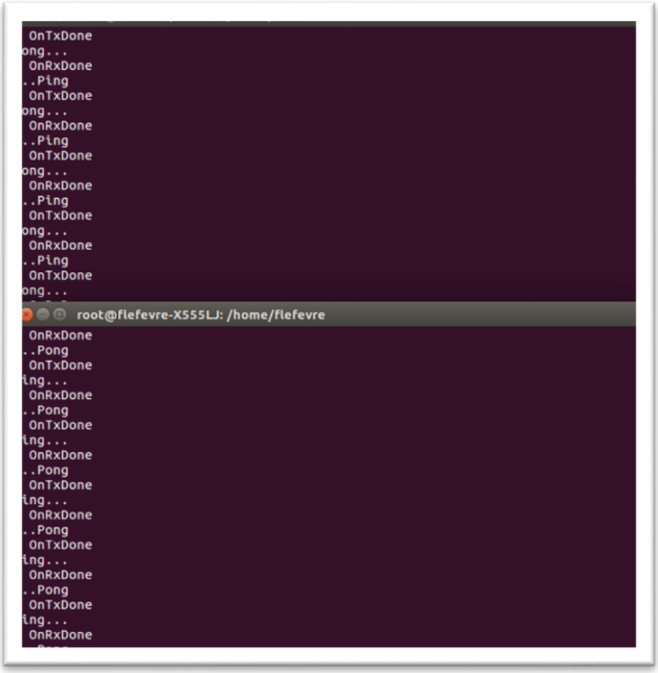

Figure 13 : Ping Pong réalisant entre deux noeuds SX1276MB1MAS

Puisque ce premier module LoRa semble fonctionner correctement, nous sommes passés au test du second type de nœud que nous allons utiliser, qui est le shield I-NUCLEO-LRWAN1 de chez ST. Cependant, après avoir compris que ce shield contenait déjà un firmware LoRaWAN et qu'il n'était donc pas possible de tester leur bon fonctionnement sans avoir un réseau LoRaWAN à disposition, nous avons remis à plus tard le test de ces modules. Pour déployer notre réseau, nous n'utiliserons qu'un seul nœud pour commencer.

Test du concentrateur

Après avoir prouvé que nos émetteurs LoRa étaient fonctionnels, nous pouvons maintenant tester la réception de paquets sur notre station de base LoRa et voir si l'on récupère bien ce que nos nœuds envoient.

Pour cela, nous avons d'abord configuré notre Raspberry Pi pour qu'elle puisse communiquer correctement par liaison SPI avec notre concentrateur. Nous avons également récupéré un script de démarrage qui doit être lancé à chaque que l'on allume notre Raspberry Pi pour que notre concentrateur soit accessible. Toutes les informations concernant cette configuration peuvent être retrouvées en suivant lien prévu à cet effet dans la bibliographie.

Une fois l'installation complétée, nous avons récupéré un ensemble de fonctionnalités permettant de tester le concentrateur aussi bien émission qu'en réception. Ces programmes proviennent du dépôt LoRa officiel de Semtech nommé « lora_gateway ». Pour notre part, on utilisera particulièrement le programme « util_pkt_logger » qui nous offre la possibilité d'enregistrer des fichiers CSV, sous forme de logs, tous les paquets reçus par la passerelle.

Nous avons donc lancé ce programme du côté de la station de base, puis nous avons utilisé le programme de ping-pong du côté du nœud pour voir si la passerelle recevait bien des trames contenant le message « PING ».

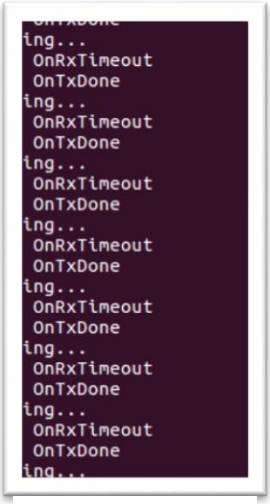

Figure 14 : Envoi de "PING" du côté émetteur

gateway ID	node MAC	UTC timestamp	us count	frequency	RF chain	RX chan	status	size	modulation	bandwidth	datarate	coderate	RSSI	SNR	payload
A555A0000000101		2018-01-18 09:43:45.793Z	3863756	867900000	0	7	CRC_OK	32	LORA	125000	SF7	4/5	-29	+8.5	50494E47-00010203-
A555A0000000101		2018-01-18 09:44:58.084Z	76154180	867900000	0	7	CRC_OK	32	LORA	125000	SF7	4/5	-35	+9.2	50494E47-00010203-
A555A0000000101		2018-01-18 09:45:01.703Z	79772468	867900000	0	7	CRC_OK	32	LORA	125000	SF7	4/5	-33	+8.2	50494E47-00010203-
A555A0000000101		2018-01-18 09:45:03.603Z	81671123	867300000	0	4	CRC_BAD	10	LORA	125000	SF7	1/2	-103	-10.5	AF7012DE-6AD378A
A555A0000000101		2018-01-18 09:45:05.322Z	83390764	867900000	0	7	CRC_OK	32	LORA	125000	SF7	4/5	-35	+9.2	50494E47-00010203-
A555A0000000101		2018-01-18 09:45:08.941Z	87009060	867900000	0	7	CRC_OK	32	LORA	125000	SF7	4/5	-35	+9.2	50494E47-00010203-
A555A0000000101		2018-01-18 09:45:12.559Z	90627348	867900000	0	7	CRC_OK	32	LORA	125000	SF7	4/5	-33	+9.0	50494E47-00010203-

Figure 15 : Trames reçues sur la passerelle

On peut alors observer que la passerelle récupère entièrement les messages envoyés par le nœud. La payload ou « données utiles » des messages contient bien la chaîne de caractères « PING » au format hexadécimal suivie de bits de padding.

Une fois que la vérification du bon fonctionnement de chaque entité du réseau a été réalisée, nous avons pu passer au déploiement du réseau LoRaWAN.

Implémentation du protocole LoRaWAN

Pour rappeler l'une des contraintes de notre cahier des charges, l'implémentation que l'on choisira ne devra pas dépendre d'un réseau public comme The Things Network (TTN). Nous sommes donc partis à la recherche d'une implémentation LoRaWAN non-officielle et open source. C'est de cette manière que nous avons vite réalisé qu'il n'y en avait que très peu de disponibles à l'heure actuelle. Effectivement, après plusieurs jours de recherche et de documentation, nous avons retenu deux solutions qui correspondaient à nos attentes.

Nous avons finalement opté pour le projet de M. Brocaar (loraserver.io) qui paraissait mieux documenté et plus ergonomique pour l'utilisateur. Les deux implémentations trouvées sont également dans la bibliographie.

Loraserver.io

Pour mettre en place notre serveur LoRaWAN, nous avons suivi toutes les indications données sur le site du développeur. Il a fallu un certain temps pour comprendre la manière dont celui-ci avait conçu le serveur mais aussi pour avoir une configuration qui correspondait à nos attentes.

Une fois que notre réseau fût administrable, il était maintenant possible d'ajouter les nœuds embarquant le firmware LoRaWAN directement sur le shield. Ce logiciel intégré permet de faire tout ce que l'on souhaite pour un nœud de notre réseau. Il fonctionne par commandes AT, c'est-à-dire que sa configuration s'effectue par liaison série en lui donnant des ordres. On envoie les commandes que l'on souhaite sur le port série pour que notre nœud ait le comportement voulu, par exemple envoie tel message avec tels paramètres radios toutes les 5 secondes. Nous avons donc flashé nos microcontrôleurs Nucleo supportant ces shields pour qu'ils envoient toutes les commandes AT nécessaires pour envoyer des trames toutes les 10 secondes. Les programmes se retrouvent sur le dépôt GIT du projet.

Après avoir terminé ces tâches et administré le réseau pour que les nœuds puissent communiquer avec le serveur, notre réseau LoRaWAN était finalement implanté.

Pour pouvoir mettre en place cette version non-officielle du serveur LoRaWAN, il y a différents services qui ont dû être installés. En effet, un premier service assurant le suivi des paquets entre la couche radio du concentrateur LoRa et la partie logicielle de notre serveur LoRaWAN est indispensable. Celui s'appelle le « packet forwarder » et permettra de transférer au serveur via UDP tous les paquets reçus par voie RF.

Un autre service également essentiel dans notre application est « Mosquitto », qui est un « MQTT broker » et implémente le protocole MQTT pour transmettre de l'information. Ce protocole-ci, basé sur TCP/IP, est un protocole de messagerie construit sur le modèle de publier-s'abonner. En d'autres termes, une entité a la possibilité de créer un sujet ou « topic » et de publier toutes sortes d'informations sur celui-ci tandis qu'une ou plusieurs autre(s) entité(s) ont l'opportunité de s'abonner à ce sujet et de recevoir toutes les informations qui en émanent. Ce principe est important puisqu'il est utilisé pour l'implémentation du serveur.

Notre implémentation quant à elle se divise en trois majeures parties :

- **Le « LoRa Gateway Bridge »** : Ce service à la base du serveur va permettre de transformer nos paquets UDP provenant du « packet forwarder » en documents JSON, un format beaucoup plus simple à traiter que des trames UDP. Ces fichiers seront via MQTT par Mosquitto directement sur le « LoRa Server ».

- **Le « LoRa Server »** : c'est la composante qui représente le « network server » au sein du réseau. Il reçoit tous les documents provenant de la passerelle et réaliser tout le traitement de ces informations ainsi que l'administration du réseau, par exemple quand un nouveau nœud veut rejoindre le réseau ou quand il faut vérifier si un nœud fait vraiment partie du réseau. C'est lui qui gère toutes les commandes MAC de manière générale. Si un message est jugé valide et conforme au protocole, il est alors transmis au « LoRa App Server ».

- **Le « LoRa App Server »** : cette dernière partie incarne le « application server » dans le réseau. En association avec le « LoRa Server », il offre la possibilité de gérer le réseau de manière interactive via une interface web. C'est sur cette interface que l'on crée des applications pour venir y ajouter des nœuds, choisir leur mode d'activation … Il est possible également d'avoir la liste de tous les paquets reçus pour un nœud en continu.

Voici un schéma de l'architecture que l'on obtient en implémentant tous les services évoqués précédemment. On notera que nous utilisons la méthode consistant à séparer le « packet forwarder » du « LoRa Gateway Bridge » et non pas celle qui rassemble les deux en une entité. De plus, tous les services présentés sont tous sur une seule et même machine, qui est notre Raspberry Pi.

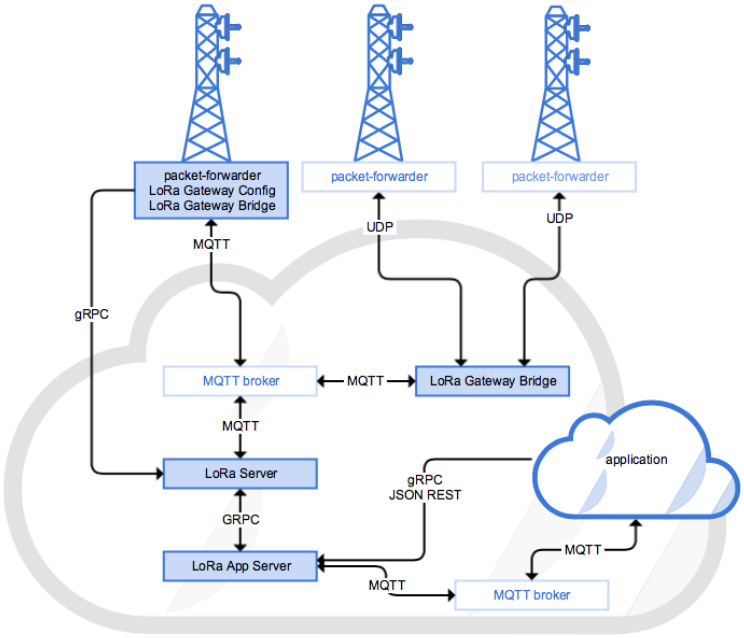

Figure 16 : Architecture de notre serveur LoRaWAN

Voici également un aperçu de l'interface web offerte par « LoRa App Server » sur laquelle nous avons pu administrer nos trois nœuds et ainsi communiquer avec le serveur. Etant donné que nous n'utilisons pas d'écran pour lancer l'interface graphique sur notre Raspberry, nous utilisons le client VNC pour pouvoir avoir cette interface graphique depuis notre PC. Une fois connecté via VNC, nous pouvons accéder à l'interface web dans un navigateur en local :

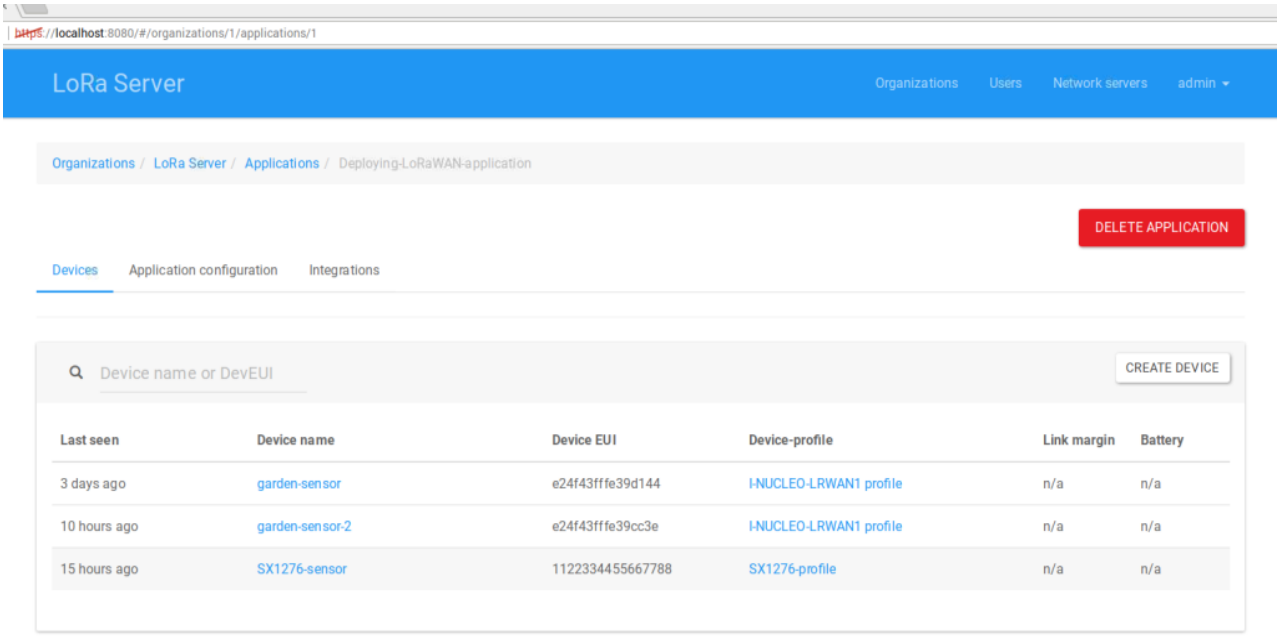

Figure 17 : Application avec nos trois noeuds

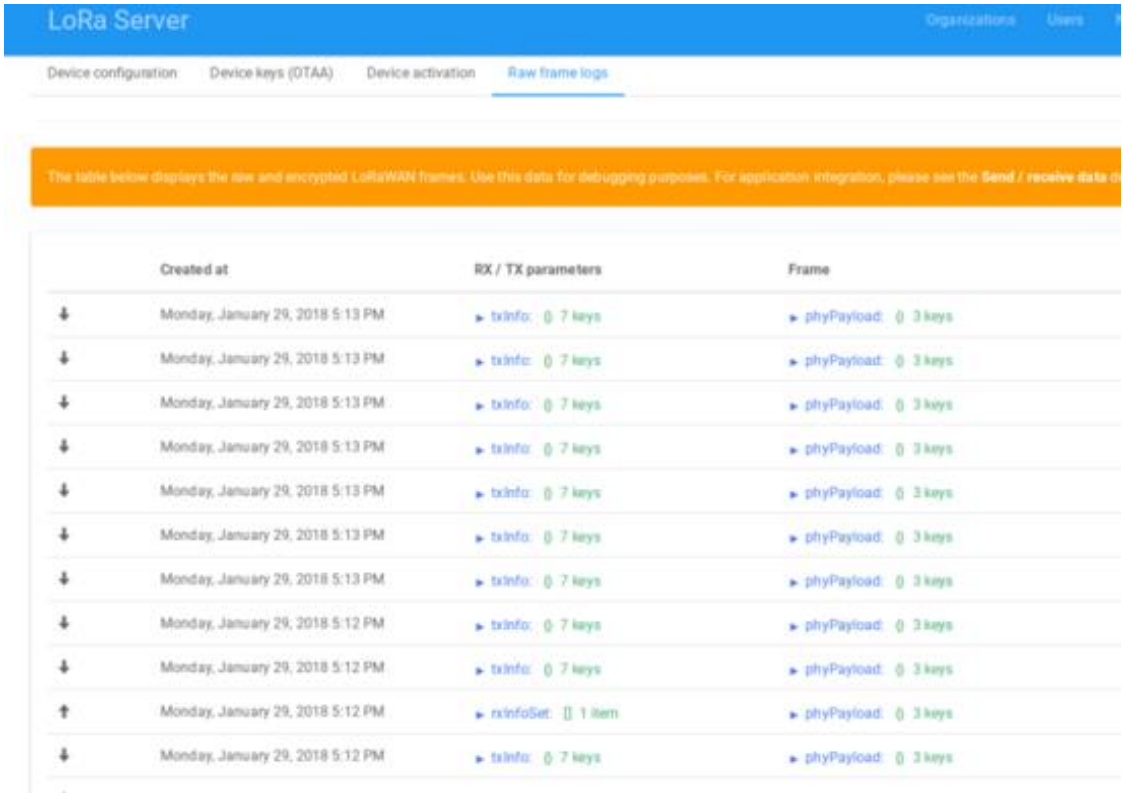

Figure 18 : Affichage des messages reçus en continu

ELK Stack

Pour compléter notre serveur LoRaWAN et pouvoir observer les données envoyées par les nœuds du réseau de manière conviviale, nous utilisons un ensemble de services que l'on appelle l'ELK Stack.

Nous avons trouvé ce projet en réalisant lorsque nous cherchions un moyen de traiter nos logs. En effet, même si l'interface web de « LoRa App Server » est grandement utile, on ne peut pas visualiser la « payload » de nos paquets qui est cryptée. De plus, on ne peut effectuer aucun traitement sur ces données. Nous souhaitions donc une application qui récupère tous les paquets reçus par le serveur LoRaWAN et nous offre la possibilité de d'interagir avec nos données pour réaliser des graphiques par exemple. L'ELK Stack semblait idéal pour cela.

Nous avons alors installé et configuré tous les services nécessaires au fonctionnement final du « stack » directement sur notre Raspberry Pi. En y ajoutant un serveur web, nous pouvons même y accéder depuis n'importe quel ordinateur à distance si l'on connait l'adresse IP de la Raspberry. Un guide pour l'installation de cette suite de services sur Raspberry Pi est fourni en bibliographie, car celle-ci est légèrement plus complexe que pour un PC classique, due en partie à son architecture ARM.

Ce projet se décompose en trois services liés entre eux, un peu de la même façon que notre implémentation LoRaWAN :

- **Logstash** : ce service, à la base de notre « stack », rassemble toutes les données que l'on souhaite exploiter au sein de notre application pour les transmettre au service suivant, qui est Elasticsearch. Logstash est capable de récupérer de l'information sous différentes formes grâce à divers plugins. L'un d'eux permet de récupérer des requêtes HTTP afin de les transmettre à Elasticsearch. Cela est essentiel pour nous car nous nous aidons de l'intégration HTTP fournie avec « LoRa App Server » pour envoyer tous les paquets reçus par le serveur via des requêtes HTTP. Nous pouvons alors récupérer ces requêtes directement avec Logstash.

- **Elasticsearch** : cette partie de l'ELK Stack va réaliser un indexage avec toute l'information récupérée de Logstash pour que celle-ci soit triée afin d'être traitée par un dernier service, qui est Kibana. C'est le maillon central du stack, que l'on peut assimiler à un cloud.

- **Kibana** : c'est la dernière pièce de notre serveur de traitement de logs qui va donner un réel intérêt à l'indexage réalisé précédemment. C'est une interface web qui va se baser sur Elasticsearch et proposer tout un tas d'outils afin d'exploiter les informations reçues, à savoir la « payload » de nos messages LoRaWAN. On a la possibilité de réaliser des graphiques et des tableaux de bord pour visualiser nos données.

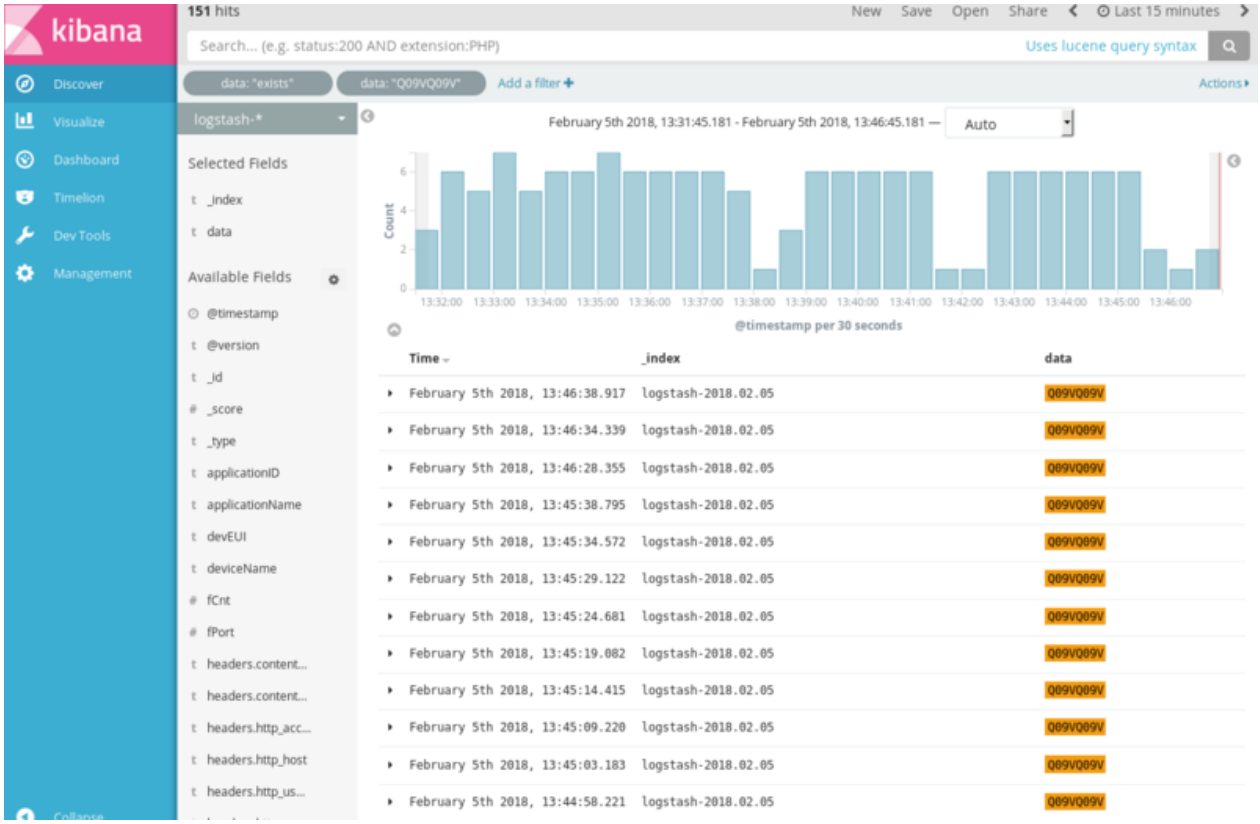

Figure 19 : Aperçu de Kibana

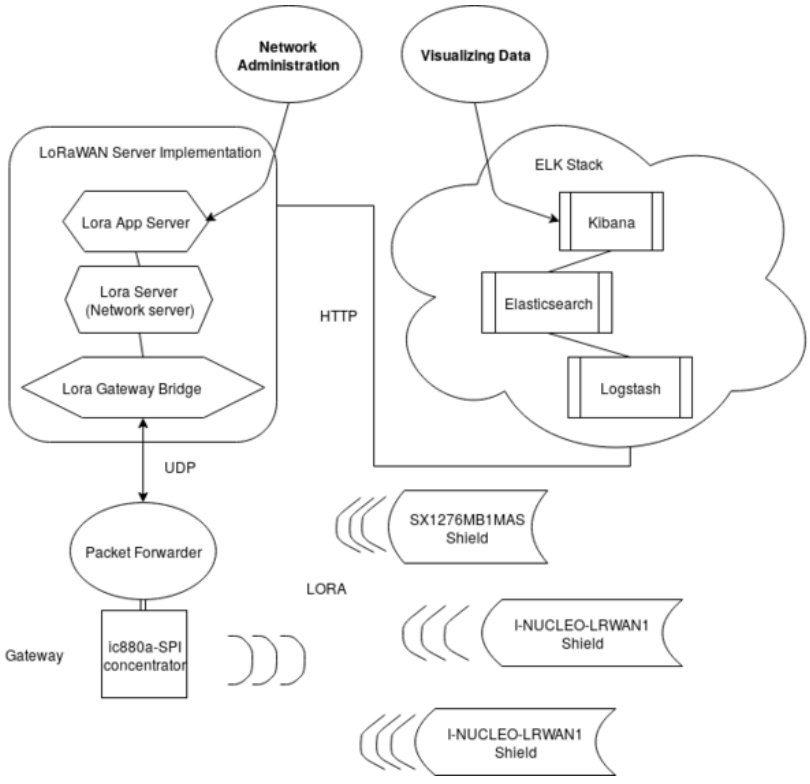

Figure 20 : Architecture finale de notre serveur LoRaWAN avec notre serveur de traitement des logs

Analyse des vulnérabilités du protocole

Après avoir obtenu un réseau fonctionnel, nous avons pu faire une liste de vulnérabilités et des faiblesses du protocole afin de développer de possibles scénarios d'attaque. Comme écrit dans le cahier des charges, nous nous sommes intéressés plus particulièrement aux attaques par rejeu. Ces attaques sont les plus pratiques à mettre en place car elles ne nécessitent pas de prendre le contrôle d'une machine du réseau. Il suffit d'avoir du matériel pour écouter le réseau et pouvoir émettre des trames quand il le faut.

Pour nous informer sur le sujet, nous avons trouvé quelques documents résumant toutes les failles que l'on peut exploiter dans LoRaWAN, en particulier une thèse de M. Yang. Nous avons également consulté entièrement la certification officielle du protocole pour connaître tous les détails de celui-ci et vérifier ainsi nos sources.

En rassemblant un maximum d'informations, nous avons pu en déduire que des attaques par rejeu étaient possibles sur notre réseau en utilisant les faiblesses de l'activation par ABP ainsi que de l'utilisation du champ Fcnt ou « frame counter » dans les messages LoRaWAN.

Nous avons vu lors de la présentation du protocole que pour une activation par ABP, les clés de session permettant le chiffrement des données étaient fixées à l'avance du côté du nœud et du serveur également. De ce fait, même si l'on souhaite générer une nouvelle session entre le nœud et le serveur, les clés de session resteront identiques. Cela signifie que deux messages provenant de deux sessions différentes avec un champ Fcnt de même valeur seront strictement identiques dans leur structure.

De plus, la gestion de ce « frame counter », justement utilisé pour éviter des attaques par rejeu, est laxiste. En effet, un message ayant un compteur d'une valeur supérieure à celle contenu dans le message précédemment reçu sera considéré comme valide par le serveur, à condition que la différence entre les deux valeurs ne dépasse pas une valeur maximale que l'on appelle « gap » et qui est définie lors de l'implémentation.

En combinant ces deux faiblesses, il devient tout à fait possible d'effectuer une attaque par rejeu. Un attaquant pourrait très bien attendre qu'une nouvelle session soit initialisée ou qu'un reset du nœud soit effectué, ce qui réinitialiserait le « frame counter » à zéro. Si cet attaquant a un moyen pour capturer tous les messages émis par le nœud, il n'aura plus qu'à retransmettre un paquet avec un « frame counter » suffisamment grand pour que l'attaque soit efficace. Une fois ce message accepté par le serveur, tous les messages suivants qui auront un compteur inférieur à celui contenu dans le paquet rejoué seront rejetés par le serveur.

Soit Cn le « frame counter » du nœud du réseau et Cm celui du message rejoué par l'attaquant :

- Si Cm – Cn <= Gap : le message rejoué par l'attaquant sera accepté. Tous les messages avec un compteur compris dans [Cn,Cm] seront rejetés.

- Si Cm – Cn > Gap : le message rejoué sera ignoré.

On notera que l'attaque la plus lourde qui pourrait être effectuée sera de rejouer un message avec un compteur Cm = Cn + Gap. Dans ce cas de figure, le réseau mettra le temps le plus long qui soit pour se remettre de cette attaque.

D'un point de vue plus technique, cette attaque utilise la méthode du « spoofing » sur le serveur, c'est-à-dire en lui envoyant un message malicieux que le serveur considère comme authentique car envoyé précédemment par le noeud "victime". Via ce procédé, on réalise ensuite un déni de service sur le nœud du réseau puisque tous les messages provenant de celui-ci ne seront plus traités par le serveur. Dans un cas où le nœud est responsable d'émettre des signaux d'alarme, cette attaque peut être dangereuse.

On notera que cette attaque est possible soit en utilisant la faille lors d'un reset pour un nœud activé par ABP, soit en attendant un « overflow » du « frame counter » puisque celui-ci est de taille limitée, à savoir 16 bits pour la version 1.0.2 de la spécification qui est implémentée sur notre réseau. Ce deuxième cas de figure peut survenir aussi bien en mode ABP qu'en mode OTAA, cependant la fréquence d'attaque est beaucoup plus faible. Nous réaliserons donc le premier cas de figure et passeront au second si jamais le premier est un échec.

Réalisation de scénarios d'attaque

Attaque par rejeu après reset

Nous avons donc commencé les phases d'attaque en se basant sur la faiblesse du mode ABP après un reset d'un nœud. Il est soit possible d'attendre reset venant d'un utilisateur du réseau ou bien de le provoquer manuellement, par exemple en mettant hors tension l'équipement si cela est possible. Voici une citation de la documentation officielle (v1.0.2) sur le protocole que nous avons exploité :

« After a JoinReq – JoinAccept message exchange or a reset for a personalized end-device, the frame counters on the end-device and the frame counters on the network server for that end-device are reset to 0. »

Ce qui nous intéresse est le reset pour un nœud activé par ABP car dans le cas d'un « join request », de nouvelles clés de chiffrement sont générées pour une nouvelle session.

Pour réaliser cette attaque, nous avons tout de même eu besoin de matériel supplémentaire. Nous avons demandé un deuxième concentrateur LoRa ainsi qu'une seconde Raspberry pour pouvoir réaliser une deuxième passerelle LoRaWAN.

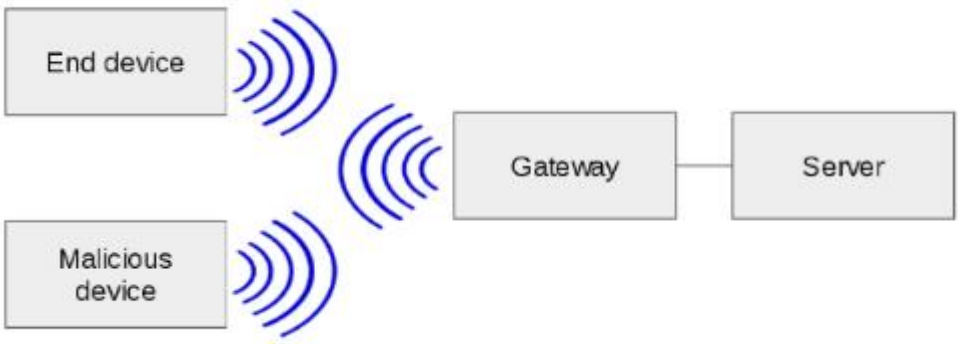

Figure 8 : Configuration réseau pour une attaque par rejeu

La configuration du réseau pour pouvoir réaliser notre attaque est présentée ci-dessus. Le « malicious device » sera dans notre cas notre seconde passerelle « pirate » ainsi que notre nœud équipé du shield SX1276 que nous avons retiré du réseau pour pouvoir s'en servir pour attaquer.

Nous avons suivi la démarche suivante pour effectuer le test d'attaque :

- Nous avons écouté en continu les paquets émis au sein de notre réseau LoRaWAN sur notre gateway pirate.
- A un instant t, nous avons manuellement provoqué le reset d'un nœud, qui initialisera du coup une nouvelle session avec un « frame counter » de retour à zéro.
- En s'aidant du décodeur de paquets LoRaWAN en ligne (mis dans la bibliographie) pour connaître le champ Fcnt d'un message, nous avons récupéré le paquet ayant le « frame counter » le plus élevé, à savoir celui précédant la nouvelle requête de « join request ». Nous avons alors utilisé notre nœud SX1276 pour réémettre tel quel le paquet en question le plus rapidement possible après le reset.
- La gateway de notre réseau devait alors normalement accepter ce paquet. A partir de ce moment, tous les paquets émis par le nœud "piraté" devraient être rejetés par la gateway jusqu'à temps que son « frame counter » dépasse celui du paquet rejoué par le pirate.

Après quelques tentatives de reset manuel, nous nous sommes aperçus d'un problème dans la gestion du « frame counter ». En effet, lors d'un reset du nœud, le compteur du côté du nœud est bien remis à zéro mais ce n'est pas le cas du côté du « network server » qui conserve la dernière valeur du compteur enregistrée. Cette réaction est tout à fait normale si l'on suit la dernière certification (v1.1.0) du protocole, qui dit la chose suivante :

« ABP devices have their Frame Counters initialized to 0 at fabrication. In ABP devices the frame counters MUST NEVER be reset during the device's life time. If the end-device is susceptible of losing power during its life time (battery replacement for example), the frame counters SHALL persist during such event. »

On se rend bien compte que notre serveur respecte donc cette règle et conserve en mémoire la valeur du compteur. Pour le nœud, nous avons une implémentation qui se réfère à la version 1.0.1 de la certification, et la valeur du compteur n'est donc jamais écrite en mémoire. En effet, dans les versions précédant la 1.1.0, le reset d'un nœud configuré en ABP doit réinitialiser le compteur du nœud, mais aussi du serveur comme énoncé dans la citation tirée de la version 1.0.2.

On peut donc conclure que cette attaque fût un échec dû au fait que le nœud et le serveur ne fonctionnent pas sur la même version de LoRaWAN, ce qui implique que chaque machine possède un comportement différent lors d'un reset.

Pour mieux comprendre la manière dont fonctionne l'implémentation LoRaWAN au niveau de la gestion de ces compteurs, nous avons contacté directement le créateur de cette application. Au bout de quelques échanges, nous avons pu comprendre que le serveur se basait sur la version 1.0.2 du protocole, mais que son créateur a corrigé les vulnérabilités en apportant quelques nouveautés de la nouvelle version 1.1.0, en partie pour éviter ces attaques par rejeu. Nous avons donc un prototype à cheval entre différentes versions de LoRaWAN, ce qui nous empêche d'exploiter réellement les vulnérabilités des versions antérieures à 1.1.0.

Vu que ce test semblait compromis, nous avons pensé à tester la seconde manière de réaliser un reset du compteur, à savoir dépasser sa taille maximale et provoquer un « overflow ». Bien que moins probable, cette attaque reste tout de même envisageable.

Attaque par rejeu après overflow

Nous allons nous intéresser ici au dépassement du compteur. Celui-ci est d'une taille de 16 bits pour la version 1.0.2 du protocole, ce qui signifie qu'un « overflow » du compteur peut se produire de manière occasionnelle, à savoir toutes les semaines si l'on envoie des trames toutes les 10 secondes. Cependant, si l'on respecte la certification LoRaWAN, un émetteur doit respecter un duty cycle de 1%, ce qui donne généralement une émission toutes les minutes ou les deux minutes, en fonction du nombre de données envoyées. Cela augmente donc la fréquence d' « overflow » du « frame counter » mais cela reste tout de même envisageable.

Pour réaliser ce test, nous allons suivre le même mode opératoire que pour l'attaque précédente. Effectivement, le but reste le même à savoir réémettre une trame au bon moment. Pour accélérer la procédure et éviter d'attendre des semaines avant le dépassement du compteur, nous allons faire en sorte que le nœud émette des trames le plus rapidement possible. En ajoutant le temps de traitement du serveur, on arrive à envoyer des messages toutes les 3 secondes environ. Cela mettrait en théorie un peu plus de 2 jours pour se produire.

Nous avons donc laissé tourner notre nœud durant quelques jours et attendu que le « frame counter » dépasse la valeur de 2^16 = 65 536. Une fois de plus, nous avons été dans l'impossibilité de réaliser cette attaque puisque la valeur du compteur ne s'est jamais réinitialisée à zéro. Le compteur continuait de s'incrémenter sans cesse.

Nous supposons donc une fois de plus que l'implémentation respecte plutôt la version 1.1.0 sur ce point plutôt que la 1.0.2 et que le « frame counter » est finalement de taille 32 bits. Voici les changements apportés sur le champ Fcnt avec la nouvelle version 1.1.0 qui justifie nos suppositions :

« FCnt changes :

- All counters are 32bits wide, 16bits not supported any more

- Separation of FCntDown into AFCntDown and NFCntDown

- Remove state synchronization requirement from NS/AS

- Remove requirement to discard frames if FCnt gap is greater than MAX_FCNT_GAP. Unnecessary with 32bit counters

- End-device Frame counters are reset upon the successful processing of a Join-Accept

- ABP device must never reset frame counters »

Nous n’avons donc pas le temps nécessaire à notre disposition pour attendre le dépassement d’un compteur de 32 bits, qui prendra beaucoup plus de temps qu’un de 16 bits bien évidemment. Cependant, nous voulions tout de même avoir une attaque de présentable.

Nous avons donc supposé que le compteur allait tout de même se réinitialiser après avoir dépassé sa valeur maximale et le remettant à zéro manuellement. Cela est donc difficilement réalisable dans la pratique puisque cela nécessite un accès physique au serveur réseau, cependant cela permet de rendre compte d’une vulnérabilité de la version 1.0.2 en respectant à la lettre la spécification et en rendant notre serveur moins intelligent qu’il ne l’est.

Nous avons remis à zéro manuellement le « frame counter » du côté du serveur puis rejoué une des trames avec une grande valeur du compteur pour que le déni de service dure le plus longtemps possible. Nous avons finalement réussir à prouver la vulnérabilité autour du « frame counter » puisque le serveur a accepté la trame rejoué et rejeté ensuite tous les messages envoyés par le nœud. Nous ne pouvons pas donner de durée concernant le déni de service complet puisque celui-ci aurait duré au-delà de la soutenance de projet.

Voici le résultat de l’attaque effectuée sur notre réseau :

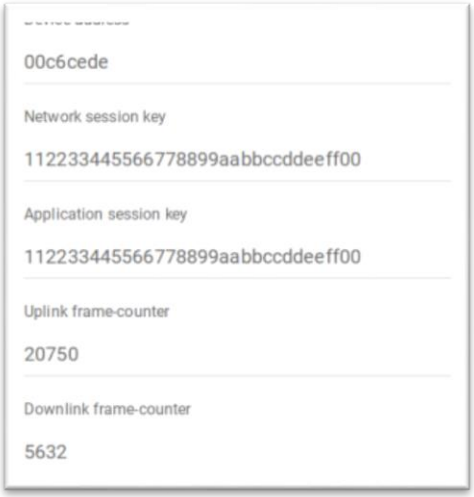

Figure 21 : Etat du "frame counter" après émission du message malicieux

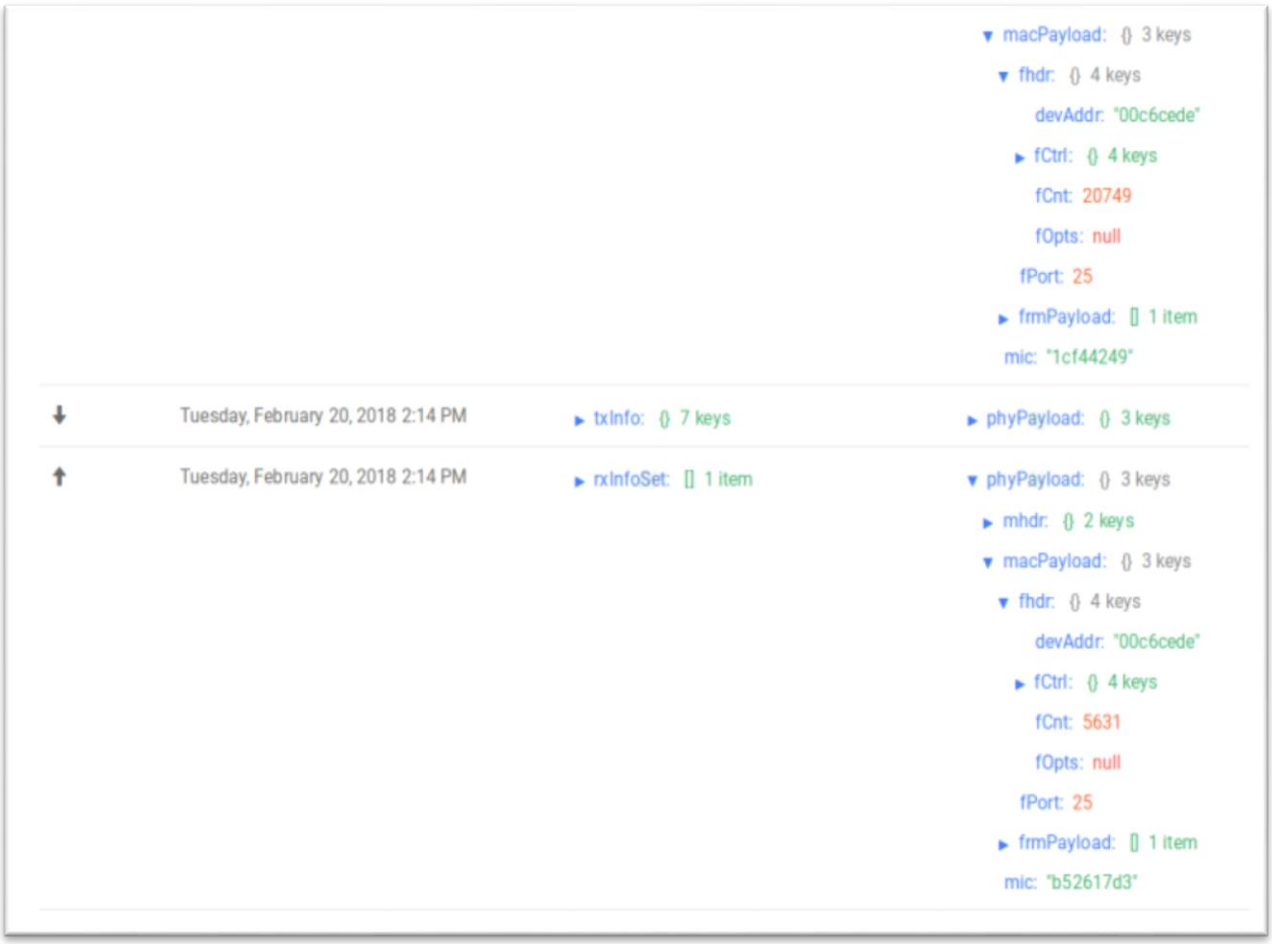

Figure 22 : Messages jugés valides par le "network server"

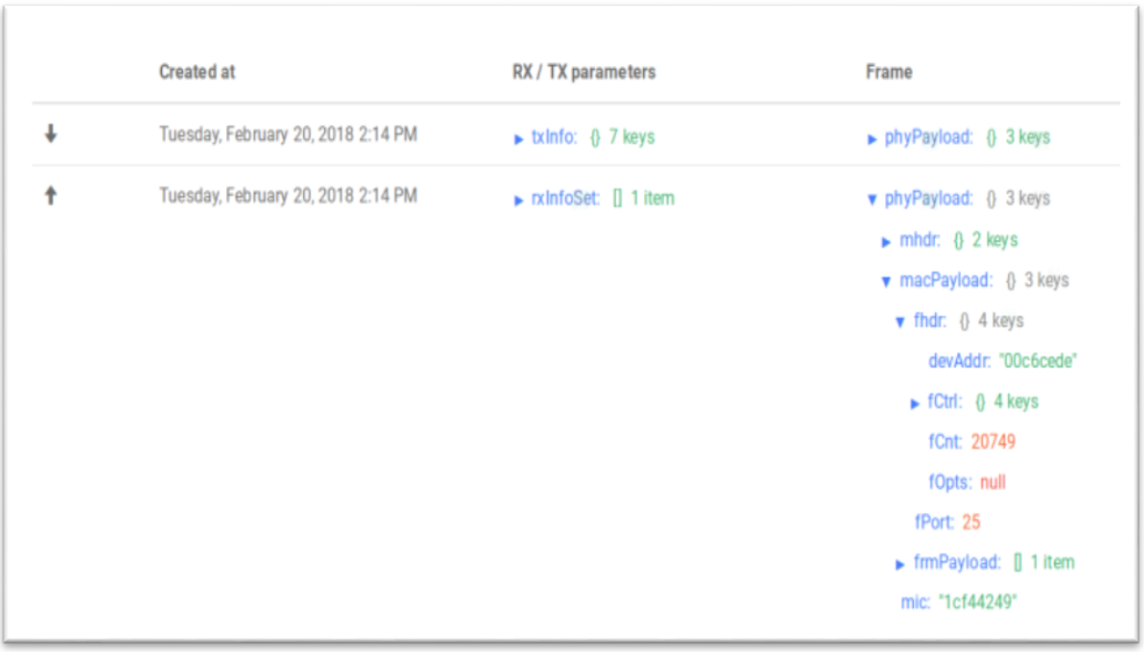

Figure 23 : Dernier message reçu

On voit bien sur la dernière image qu'après avoir reçu le message contenant une valeur de 20 749 comme « frame counter », plus aucun message n'est reçu depuis ce moment. Ce déni de service pourrait durer plusieurs semaines en fonction de l'application.

Autres attaques possibles

Il existe également d'autres vulnérabilités dans la version 1.0.2 de LoRaWAN qui sont connues et peuvent être exploitées, mais nous ne nous y sommes pas intéressés. La plupart se base également sur la mauvaise gestion du « frame counter » mais demande parfois d'avoir le contrôle sur un équipement du réseau.

Il est possible de réaliser du « ACK spoofing », c'est-à-dire utiliser une passerelle pirate et se faire passer pour la passerelle du réseau pour faire croire au nœud que les messages qu'il envoie sont validés en lui envoyant des « acknowledgments ». Les messages ne sont en réalité jamais arrivés jusqu'à la passerelle du serveur. Nous avons aussi la technique du « eavesdropping » qui va écouter le trafic du réseau pour faire des analyses statistiques en réalisant des paires de paquets pour en déduire les clés de chiffrement. Une dernière possibilité serait de faire du « bit flipping » sur les messages transitant entre le serveur réseau et le serveur applicatif puisque la vérification du MIC se fait au niveau du serveur réseau. Rien n'empêche de modifier le contenu d'une trame après cette vérification.

Problèmes rencontrés

Au cours de ce projet, nous avons fait face à de nombreux problèmes pour lesquels nous n'avons parfois pas vraiment d'indications. Le sujet étant en grande partie de la recherche, il arrive plus souvent d'être livré à soi-même comparé à du développement comme la création d'une application mobile par exemple. De plus, le protocole LoRaWAN est une technologie plutôt récente et donc peu de documentation est disponible. Il est parfois arrivé de passer des heures à chercher une solution à une erreur et de ne rien trouver de pertinent.

Concernant la conception du shield, nous avons dans un premier temps eu du mal à comprendre pourquoi notre carte ne fonctionnait pas. Nous avons pensé à plusieurs raisons avant de se rendre compte que le problème venait du fait que la carte n'était pas métallisé, et ne pouvait donc pas conduire le courant entre les couches. Pour ma part, c'était la première que je réalisais une carte multicouche et je n'avais donc jamais été confronté à ce souci. Après avoir soudé directement sur les pistes de fin fils métalliques comme décrit dans le rapport, tout cela s'est arrangé.

Au début de ce projet, j'avais également du mal à distinguer la différence entre LoRa, la modulation électronique utilisée pour l'envoi de messages, et LoRaWAN qui est globalement un protocole réseau. En effet, nous avions commencé par déployer l'ELK Stack en premier lorsque nous devions trouver une implémentation pour notre serveur LoRaWAN. Nous pensions qu'en utilisant juste le « packet forwarder » ainsi qu'un serveur nous permettant de traiter les logs reçus, il aurait été possible de mettre en place un réseau LoRaWAN. Nous avons compris par la suite qu'un vrai serveur implémentant la certification était indispensable pour gérer les équipements du réseau et pas juste pour traiter les informations reçues. Finalement, nous avons lié notre serveur LoRaWAN à l'ELK Stack.

Nous avons eu également beaucoup de difficultés pour appliquer les scénarios d'attaque envisagés sur notre réseau car l'implémentation choisie n'est pas vraiment fixée sur une version de LoRaWAN, elle est à cheval sur différentes versions pour corriger les faiblesses de l'une d'entre elles. Effectivement, beaucoup d'attaques vérifiées sur la version 1.0.2 n'ont pas pu être appliquées car notre implémentation a corrigé ces vulnérabilités. Cela est de bonne augure pour un utilisateur qui veut déployer son propre réseau, mais pas pour un attaquant qui souhaite utiliser ces failles.

Il est aussi vrai que la spécification LoRaWAN, bien que très détaillée, est parfois ambiguë sur certains points ce qui fait que l'on peut interpréter différemment certaines notions. C'est ce qui fait que les implémentations non officielles peuvent être différentes sur certains points.

Perspectives pour le projet

Bien que le cahier des charges ait été rempli en majeure partie, il est tout de même possible de donner des suites à ce projet plus proche d'un cas réel ou d'être plus élaboré.

Il pourrait être intéressant de tenter à nouveau des attaques par rejeu de manière plus poussée car nous n'avons pas vraiment eu le temps de réaliser des tests approfondis. Nous souhaitions juste prouver une faille dans le protocole. Il existe toujours la possibilité qu'il y réellement une faille au niveau du « frame counter » sans devoir le réinitialiser manuellement.

Au cours de ce projet, nous nous sommes intéressés au cas des attaques par rejeu. Cependant, il reste tout de même d'autres types d'attaques qui peuvent être testés, comme le « ACK spoofing ». La plupart d'entre elles sont énoncées dans la partie associée aux attaques.

Nous avons aussi fait le choix de travailler avec l'implémentation LoRaWAN de M. Brocaar à l'étape du déploiement du réseau. Bien que celle-ci fonctionne correctement, nous en avions trouvé une seconde que nous n'avons pas du tout testée. Il serait judicieux d'utiliser cette implémentation pour vérifier si les attaques non réussies sur la première implémentation sont réalisables sur la seconde. Effectivement, cette dernière se réfère également à la version 1.0.2 du protocole. Il y a alors possibilité de vulnérabilités.

Tout au long de notre travail, nous ne nous sommes pas réellement intéressés au design des équipements de notre réseau, mise à part le shield qui permet de rendre plus compact une passerelle. Il aurait été possible de réaliser des boîtiers pour permettre d'implanter notre réseau dans un cas réel.

A la fin du projet, nous avions comme possible mission de réaliser un sniffer de paquets LoRaWAN qui doit réémettre chaque trame reçue pour évaluer la manière dont un serveur gère la redondance ou le rejeu. La réalisation d'un sniffer plus intelligent pourrait être envisagée, dont le but serait de détecter automatiquement le reset d'un nœud du réseau et d'émettre une trame avec une grande valeur de « frame counter » parmi celles qu'il enregistre pour attaquer automatiquement.

Conclusion

Tout au long de ce mémoire, nous avons pu synthétiser toute notre période de projet de fin d'études. Nous avons commencé par rappeler le cahier des charges de notre sujet avec les objectifs et le matériel emprunté. Deuxièmement, nous avons présenté la spécification LoRaWAN pour que ce protocole devienne plus familier. Après avoir assimilé la théorie, nous avons évoqué toutes les tâches effectuées au cours des dernières semaines, en passant du déploiement du réseau aux phases d'attaques sur celui-ci. Pour conclure, nous avons listé tous les difficultés rencontrées lors du projet et donné plusieurs perspectives pour continuer sur ce travail.

Nous pouvons considérer que le cahier des charges initial a été rempli avec plus ou moins satisfaction. En effet, il aurait été préférable de réaliser plus de scénarios d'attaques sur notre réseau après l'avoir déployé. Cependant, nous avons passé un temps plutôt conséquent pour implanter ce réseau, ce qui nous a laissé beaucoup moins de temps que prévu pour les attaques. De plus, nous n'avons pas réussi une attaque qui pourrait fonctionne forcément sur un réseau LoRaWAN public. Nous avons dû adapter la configuration de notre réseau pour rendre notre attaque possible.

Ce projet aura été une bonne initiation à la recherche et m'a permis d'approfondir un peu plus mes connaissances en réseau et sécurité informatique, domaine dans lequel je voulais avoir une expérience, que ce soit un stage ou un projet. J'en retiendrais que c'est un domaine où les enjeux sont très motivants d'un point de vue d'éthique et qu'il nécessite une grande détermination pour arriver à ses fins. En effet, c'est un milieu axé sur la recherche où l'on est confronté à des cas peu ou pas connus. Il est possible de passer des jours à réaliser des tests qui se révéleront par la suite infructueux, ce qui peut s'avérer frustrant.

Ce qui est appréciable de travailler sur l'internet des objets, c'est que l'on prend connaissance de technologies qui risquent d'être fortement utilisées dans le futur pour de nombreuses applications et sur lesquelles tout le monde n'est pas forcément au point, ce qui est un atout pour notre bagage technique. Désormais, je pourrais aisément, en ayant le matériel nécessaire à disposition, déployer mon propre réseau LoRaWAN mais assi le défendre et ainsi réaliser ma propre application IoT, en prenant comme exemple la domotique pour sa propre maison.

Bibliographie

Etat de l'art du protocole LoRaWAN et illustrations :

https://fr.wikipedia.org/wiki/LoRaWAN

https://www.lora-alliance.org/

https://www.frugalprototype.com/technologie-lora-reseau-lorawan/

http://www.ioi-labs.com/technologies/comprendre-reseau-lorawan7

Sources des fichiers de design pour la conception du shield :

https://www.tindie.com/products/gnz/imst-ic880a-lorawan-backplane-kit/

Code source pour le programme de ping-pong :

https://os.mbed.com/users/dudmuck/code/SX1276PingPong/

Guide pour la configuration de la Raspberry Pi avec le concentrateur :

https://wireless-solutions.de/images/stories/downloads/Radio%20Modules/iC880A/iC880A-SPI_QuickStartGuide.pdf

Dépôt regroupant les fonctionnalités pour tester le bon fonctionnement de la gateway :

https://github.com/Lora-net/lora_gateway

Implémentation LoRaWAN de M. Brocaar que nous avons choisie :

https://www.loraserver.io/

Seconde implémentation trouvée qui peut être intéressante à étudier :

https://github.com/gotthardp/lorawan-server

Dépôt contenant le programme de suivi de paquets :

https://github.com/Lora-net/packet_forwarder

Documentation sur les vulnérabilités du protocole LoRaWAN :

https://repository.tudelft.nl/islandora/object/uuid:87730790-6166-4424-9d82-8fe815733f1e/datastream/OBJ/download (thèse de M. Yang)

https://labs.mwrinfosecurity.com/assets/BlogFiles/mwri-LoRa-security-guide-1.2-2016-03-22.pdf

https://lirias.kuleuven.be/bitstream/123456789/587540/1/camera_ready.pdf

https://medium.com/@brocaar/notes-on-lorawan-security-7e741a8ee4fa

Certification LoRaWAN officielle

Décodeur de paquets LoRaWAN :

https://lorawan-packet-decoder-0ta6puiniaut.runkit.sh/

Référence officiel de l'ELK Stack :

https://www.elastic.co/fr/elk-stack

Guide d'installation de l'ELK Stack sur la Raspberry Pi :

https://thesecuritystoic.com/2017/08/home-security-iii-elk-on-a-raspberry-pi/