Apprentissage automatique pour la détection d'attaques par déni de services

par

Robin CAVALIERI
PFE - IMA5

Département **I**nformatique - **M**icroélectronique - **A**utomatique

Tuteur : Thomas VANTROYS

Sommaire

I - Mise en contexte

Les attaques par déni de service ont pour objectif de rendre indisponible un service en surchargeant le trafic sur un réseau ou encore en perturbant les connexions entre un émetteur et un récepteur.

Ces types d'attaques sont très répandus sur de nombreux réseaux notamment dans l'Internet of Things, où certains objets connectés s'avèrent être vulnérables.

L'objectif de ce projet est d'être capable d'identifier les différents types d'attaques DoS pour ensuite enregistrer un trafic représentatif :

- de risques
- normal

pour finalement réaliser un apprentissage automatique de ces cas à travers un algorithme intelligent. Cet algorithme viendra classer un flux de données selon des caractéristiques propres au réseau (ping, puissance...).

Le réseau sur lequel ce projet est basé est un réseau LoRa munit d'un récepteur, d'un émetteur classique et d'un émetteur pirate.
Les données envoyées seront des températures générées de façons aléatoires et intégrées dans des trames.

Nous sommes donc amenés à créer un programme capable d'écouter le trafic par port série, de le traiter et d'extraire ses caractéristiques pour ensuite évaluer si celui-ci représente une menace ou non. L'interface homme / machine sera une application web avec affichage des données en temps réel et avertisseurs visuels en cas d'attaque.

Nous vous souhaitons une bonne lecture.

II - Matériel

Pour réaliser ce projet, nous disposons du matériel suivant :

Nom	Quantité	Description
STM32 ARM	3	Microcontrôleurs pogrammables par ST : http://www.st.com/en/microcontrollers/stm32-32-bit-arm-cortex-mcus.html
Modules LoRa SX1276	3	Émetteurs / Récepteurs longue portée par Semtech : https://www.semtech.com/products/wireless-rf/lora-transceivers/SX1276

Le module LoRa est directement pluggable sur la carte STM32. Ainsi, il n'est pas nécessaire de réaliser de PCB pour créer un shield.

Programmation des microcontrôleurs

Concernant la programmation et le flashage de la STM32, nous utilisons le compilateur en ligne de Mbed disponible au lien ci-dessous :
https://os.mbed.com/compiler/

L'avantage est que Mbed fournit directement ses librairies pour l'utilisation des modules LoRa :
https://os.mbed.com/users/GregCr/code/SX1276Lib/docs/tip/classRadio.html

Pour programmer les émetteurs et le récepteur, nous importons la bibliothèque ci-dessus dans un projet hébergé sur la plateforme Mbed. Toutes les bibliothèques sont en C++. Le compilateur génère un fichier *.bin* qui permet de flasher directement le microcontrôleur en versant le fichier à la racine de celui-ci via liaison USB.

Environnement de développement

Le langage de développement pour la partie back est le Python. Python fournit des librairies faciles à implémenter notamment en matière de réseaux de neurones avec *Keras*, une implémentation de *Tensorflow* : une librairie IA créée par Google. Les matrices de données sont également facilement manipulables avec *Numpy* et *Pandas*.

l'IDE utilisé est *Spyder*. Il fait partie de l'environnement Anaconda où les librairies sont téléchargées et installées à partir de l'*Anaconda Prompt*, une console dédiée :
tutoriel d'installation

La partie front est, elle réalisée à partir d'une base de HTML incluant du javascript. Un serveur *Apache2* héberge notre page web. Ce serveur a été installé à partir de XAMPP, un utilitaire sous Windows :
https://www.apachefriends.org/fr/index.html

III - Le réseau LoRa

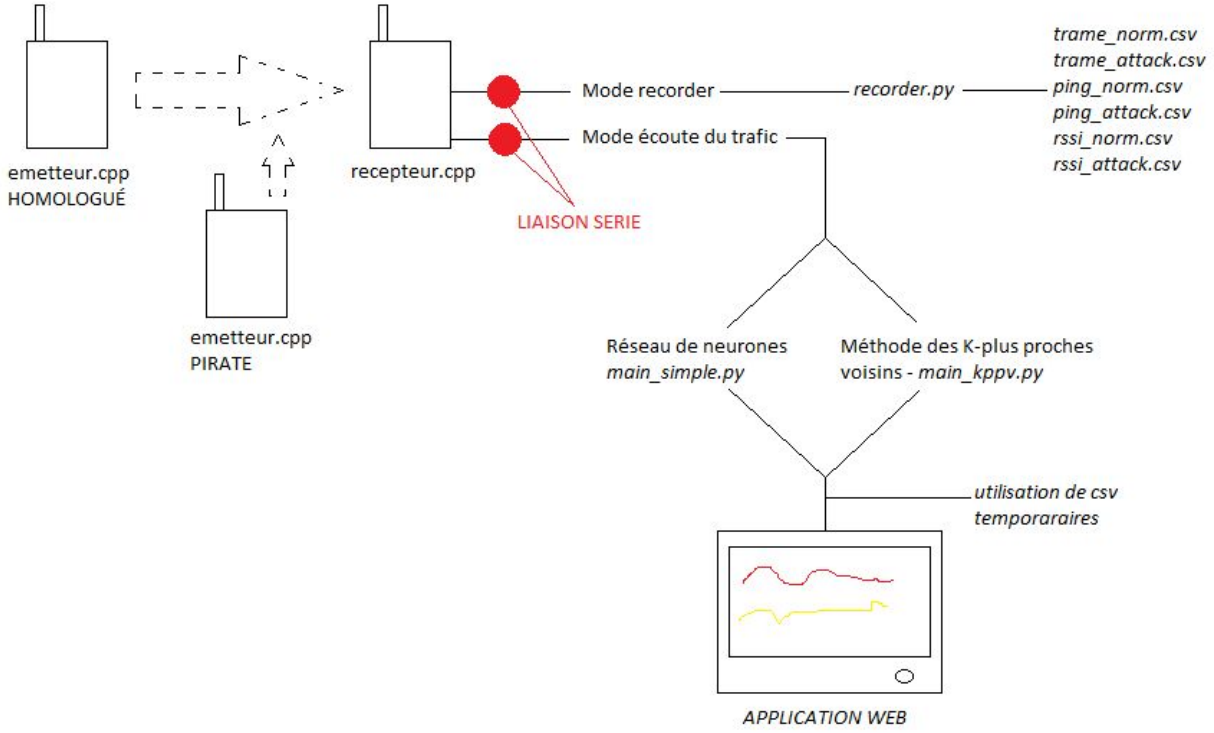
Le réseau LoRa (Long Range) est un réseau adapté à l'IoT puisqu'il permet de transmettre uniquement des petits paquets allant de 0,3 à 50 kb/s. Il n'est donc plus nécessaire d'employer des réseaux comme de la 3 ou 4G pour faire transiter ce type de données.

Ici, dans notre cas, il est adapté à l'envoi de températures sur 2 voire 3 octets par exemple. La longue portée est son principal atout. Contrairement aux IoT fonctionnant dans un réseau Bluetooth ou WiFi, il est possible ici d'émettre jusqu'à plusieurs kilomètres (1 à 20 selon le bruit environnant) grâce à la modulation d'ondes radio.

Nous tenons à préciser que le réseau LoRa n'est pas le sujet de cette étude mais uniquement un moyen d'émettre et de recevoir des trames.

IV - Architecture logicielle

1 - Schéma de principe



2 - Explications

Un récepteur reçoit des données de la part d'un émetteur. Ici, les données sont les trames contenant des températures. En utilisant le fichier *recorder.py*, il est possible d'enregistrer les données jugées importantes pour comprendre et interpréter le trafic à savoir :

- les trames
- le ping
- la puissance de réception : RSSI

Chaque partie du code est exécutable indépendamment des autres en réalisant une sélection puis un CTRL+Enter. Ainsi, si vous souhaitez réenregistrer qu'une seule partie de votre dataset, il vous est plus simple et plus rapide de passer par cette méthode. Toutes les données sont ensuite exportées au format csv. Elle seront réutilisées dans la partie qui va suivre.

En écoutant le trafic, il est possible à partir du dataset, de réaliser des prédictions quant au comportement des trames reçues.
Le réseau de neurones utilisé dans *main_simple.py* tire son apprentissage du fichier *ANN.py*. Cela évite encore une fois de recompiler un code et de réaliser à nouveau des apprentissages. Ainsi, dans le *main_simple.py*, seules les prédictions sont réalisées.

main_kppv.py joue le même rôle mais en utilisant la méthode des K-plus-proches voisins pour classer nos données et prédire l'état du trafic. Cette méthode sera expliquée plus en détails dans la stratégie.

Une application web d'une seule page vient renforcer le système de détection en utilisant des données présentes dans des fichiers csv temporaires et en les traitant.

V - Stratégie

1 - Émission de trames

À partir du projet PING-PONG fournit par Mbed (présent sur le git), nous sommes parvenus à envoyer des trames. Le modèle choisi pour l'envoi est le suivant :

```
//Converti en ASCII
testMsg[0] =  0x7C; //emetteur |
testMsg[1] =  0x46; //emetteur F
testMsg[2] =  0x33; //emetteur 3
testMsg[3] =  0x30; //emetteur 0
testMsg[4] =  0x33; //emetteur 3
testMsg[5] =  0x52; //emetteur R
testMsg[6] =  0x45; //emetteur E
//
testMsg[7] =  0x46; //recepteur F
testMsg[8] =  0x30; //recepteur 0
testMsg[9] =  0x39; //recepteur 9
testMsg[10] =  0x31; //recepteur 1
testMsg[11] = 0x43; //recepteur C
//
testMsg[12] = dizaines_c; //temperature 2 dizaines
testMsg[13] = unites_c; //temperature 0 unités
testMsg[14] = 0x7C; //emetteur |
//
testMsg[15] = centaines_cpt;
testMsg[16] = dizaines_cpt;
testMsg[17] = unites_cpt;
```

[0] : caractère de début de trame
[1…6] : identifiant de l'émetteur
[7…11] : identifiant du récepteur
[12…13] : température
[14] : caractère de fin de trame
[15…17] : compteur de trame

L'objectif était dans un premier temps d'être capable de savoir si oui ou non l'émetteur était connu du récepteur. Le caractère de début et de fin de trame : "|" est utile pour déterminer si des octets ont été perdus. Enfin, le compteur de trame, lui, permet de garder un oeil sur le trafic et prévenir des rejets et des sauts de trame. Ce compteur va de 1 à 100 car nous estimions ce nombre d'itérations suffisant au contrôle du trafic.

```
Radio.SetTxConfig( MODEM_LORA, TX_OUTPUT_POWER, 0, LORA_BANDWIDTH,
                   LORA_SPREADING_FACTOR, LORA_CODINGRATE,
                   LORA_PREAMBLE_LENGTH, LORA_FIX_LENGTH_PAYLOAD_ON,
                   LORA_CRC_ENABLED, LORA_FHSS_ENABLED, LORA_NB_SYMB_HOP,
                   LORA_IQ_INVERSION_ON, 4000 );
```

fonction transmission : settings

```
Radio.Send(Buffer, BufferSize);
```

fonction d'envoi

Ci-dessus se trouvent deux fonctions importantes à la compréhension du fonctionnement général. Le programme d'envoi possède de nombreuses constantes qui viennent représenter les paramètres de transmission comme la fréquence (868 MHz), l'étalement spectral ou encore la puissance de sortie du signal.

Les facteurs sur lesquels nous avons travaillé sont :
- la puissance de sortie
- la suppression de l'étalement en fréquence
- la taille du buffer
- le canal de communication
- la taille de la trame

Le modèle de Mbed est un programme fonctionnant avec des interruptions. De nombreuses fonctions sont donc restées identiques car fournies de la sorte par la librairie. L'envoi est ensuite réalisé dans un main avec une boucle infinie et un timer à chaque tour pour maîtriser la fréquence d'émission.

2 - Réception de trames

La réception des trames suit un modèle identique à la transmission.

```
Radio.SetRxConfig( MODEM_LORA, LORA_BANDWIDTH, LORA_SPREADING_FACTOR,
                   LORA_CODINGRATE, 0, LORA_PREAMBLE_LENGTH,
                   LORA_SYMBOL_TIMEOUT, LORA_FIX_LENGTH_PAYLOAD_ON, 0,
                   LORA_CRC_ENABLED, LORA_FHSS_ENABLED, LORA_NB_SYMB_HOP,
                   LORA_IQ_INVERSION_ON, true );
```

fonction reception : settings

```
Radio.Rx(RX_TIMEOUT_VALUE); //Reception mode en continu car TIME_OUT_VALUE = 0
```

fonction d'envoi

La constante RX_TIMEOUT_VALUE est posée à 0 pour recevoir en "continu" les données présentes dans l'air. L'émetteur et le récepteur sont paramétrés sur la même fréquence et sur le même canal. Le modèle d'envoi est reproduit et toutes les fonctions de réception et de traitement sont appelées dans le main qui contient lui même une boucle infinie.

Une trame est traitée toutes les 5 secondes selon un trafic "normal". Elle est renvoyée sur le port série de ma STM32 avec la puissance de réception. Pour qu'une trame soit renvoyée ainsi, le buffer doit contenir les mêmes premiers caractères que ma trame type déclarée au chapitre précédent. Pour cela nous utilisons la fonction *strcmp* de la librairie *string.h*. Un paramètre correspondant à la longueur de la trame moins le nombre de variables est ajouté à la fonction. Ainsi nous ne comparons que le caractère de début de trame, l'ID de l'émetteur et celui du récepteur.
C'est en quelques sortes un moyen de protéger le système de traitements inutiles pouvant venir de trames pirates par exemple. C'est également une condition utile si

plusieurs émetteurs réalisent un envoi à plusieurs récepteurs cibles. L'envoi en broadcast nécessite que chaque récepteur vérifie si le message lui est adressé.
Les messages seront écrits sur le port série de la sorte :

```
//Buffer de données
debug("%s\n",(char*)Buffer);
//Puissance de reception du signal
debug("%d\n",RssiValue);
```

La trame s'affiche sur la ligne i et la puissance sur la ligne i+1. la vitesse de communication est de 9600 bauds. Ensuite tout est récupéré par le programme python pour le traitement des données reçues.

3 - Simulation d'attaques

La simulation d'attaques et basée sur trois paramètres :

- Le numéro de trame
- Le ping
- La puissance d'émission (ici captée par le récepteur)

Le numéro de trame sert à vérifier si aucun rejet n'est effectué. En effet, recevoir une trame avec le même numéro dans un laps de temps normalement accordé à la réception de 2 trames différentes pourrait être synonyme d'attaque.

Le simulateur de trafic pour les tests est le suivant :

NORMAL	ATTAQUE	MÉTHODE
ping = 5000 ms	ping = 1000 ms	utilisation d'un timer "wait_ms()" à chaque tour de boucle permet de contrôler la fréquence d'émission des trames dans le programme emetteur.cpp
rssi = 14 dBm	rssi = 20 dBm	Modification de la constante TX_OUTPUT_POWER dans le programme emetteur.cpp

Au départ, Nous travaillions sur des envois bien plus rapides de l'ordre de 500 ms - 100 ms. Nous avons augmenté la durée du timer afin d'être en mesure de mieux observer les phénomènes, de ne pas être limité par la liaison série et surtout d'être en mesure de réaliser des acquisitions correctes pour l'apprentissage. De plus avec une fréquence d'envoi élevée, les tests sur réseau de neurones s'avéraient être difficiles à réaliser car le temps de calcul était supérieur au ping (tests détaillés au chapitre suivant).

Pour simuler les rejets, nous n'incrémentons plus le compteur de trames et renvoyons donc le même contenu plusieurs fois d'affilée. Une condition vient vérifier si l'ordre des trames est respecté ou non.

4 - Traitement des données

À la sortie du port série, un traitement est données est réalisé via les programmes python afin qu'elles soient enregistrées ou étudiées. Le port série est écouté à partir de la méthode *readline()* fournie par la librairie *serial.* Pour utiliser le code sur Linux, il vous faudra uniquement changer le nom de port déclaré sur ma machine comme 'COM4'.

a) Enregistrement des données

L'enregistrement des donnée sest réalisé dans le fichier *recorder.py*. Ce code permet d'enregistrer la puissance, les trames brutes et le ping. Chaque valeur est associée à sa classe : 0 pour un trafic normal et 1 pour un trafic potentiellement malveillant (APPRENTISSAGE SUPERVISÉ : nous connaissons la classe des éléments du dataset).
Le dataset est donc composé de 6 fichiers csv de 3000 individus chacun (initialement) :

- trame_norm.csv
- trame_attack.csv
- ping_norm.cav
- ping_attack.csv
- rssi_norm.csv
- rssi_attack.csv

Pour enregistrer les trames, nous vérifions si l'entête correspond au modèle défini dans le programme d'emission. Cette fonction booléenne est appelée *record_condition* et prend en argument la ligne du port série. A chaque fois que la condition est respectée, la trame et sa classe sont enregistrées de la sorte :

```
#Condition d'enregistrement
if(record_condition(x)==True):
    writer.writerow(('1', x[0:TRAME_SIZE]));
    #writer.writerow(('2', x[0:TRAME_SIZE]));
    nb_individus = nb_individus + 1;
```

Au niveau de l'enregistrement des puissances de réception, la procédure est pratiquement identique. Seulement, il faut lire le seconde ligne. nous faisons donc deux appels à la fonction *readline()*. Une conversion ASCII - décimale est réalisée avant l'ajout d'une ligne dans notre fichier csv .

Le ping est enregistré (en milliseconde) selon un chronomètre placé entre l'arrivée de deux trames types (*record_condition*). Nous enregistrons donc une population de 3000 individus à compter de la seconde trame. Une suppression des caractères inutiles comme les "\n" est également réalisée pour éviter d'avoir à retravailler les données avant apprentissage.

b) Étude des données de puissance

Le RSSI est une valeur qui fluctue fortement fonction de l'environnement. C'est pourquoi il a été difficile d'établir un dataset correspondant à la puissance. Nous avons d'abord réalisé une tentative d'étude et d'apprentissage sur une différence de puissance de réception entre la trame i et la trame i+1. Cependant, si des trames sont menaçantes, notre algorithme ne le détectera qu'à la seconde trame d'attaque reçue. Une solution serait de compléter cette détection avec des conditions additionnelles pour éviter les avertissements abusifs ou inexistants.

Ensuite, une seconde approche a été l'enregistrement de la puissance de réception comme dataset. Cette méthode a été préférée à la première car elle nécessite moins de calculs et est tout aussi efficace. En effet, une fois normalisées, les valeurs de puissance offrent un dataset hétérogène et donc permettent la bonne détection d'attaque.

5 - Apprentissages supervisés

Les apprentissages supervisés consistent à faire apprendre à la machine qu'une entrée correspond à une classe. Les méthodes par réseau de neurones et par KPPV ici utilisées correspondent à des algorithmes supervisés.

a) Architecture des matrices d'apprentissage

C'est une des étapes clefs du projet de classification. Comment organiser ses matrices pour apprendre ? Diminuer le temps d'exécution ?
Dans un réseau de neurones, les entrées doivent correspondre à une ligne de la matrice :

Ping	Puissance	Target (0,1)

Seules les deux premières colonnes sont passées en entrée. La dernière est utile lors de l'apprentissage et de la phase de test.

Dans le cas d'un réseau de neurones, les matrices sont normalisées aux valeurs de la fonction d'activation soit dans notre cas entre 0 et 1.
Pour cela, nous appliquons à toutes les valeurs de la matrice :

$$val_{norm} = \frac{max - val}{max - min}$$

min : valeur minimale de la matrice
max : valeur maximale de la matrice

Pour les K-plus proches voisins, les données sont agencées autrement. La méthode requiert 2 matrices :

- une pour le ping
- une pour la puissance

Chacune d'entre elles doit contenir :

- Une valeur
- La classe à laquelle cette valeur appartient
- la distance séparant les valeurs de l'objet à classer

Classe (0, 1)	Ping	Dist. euclidienne

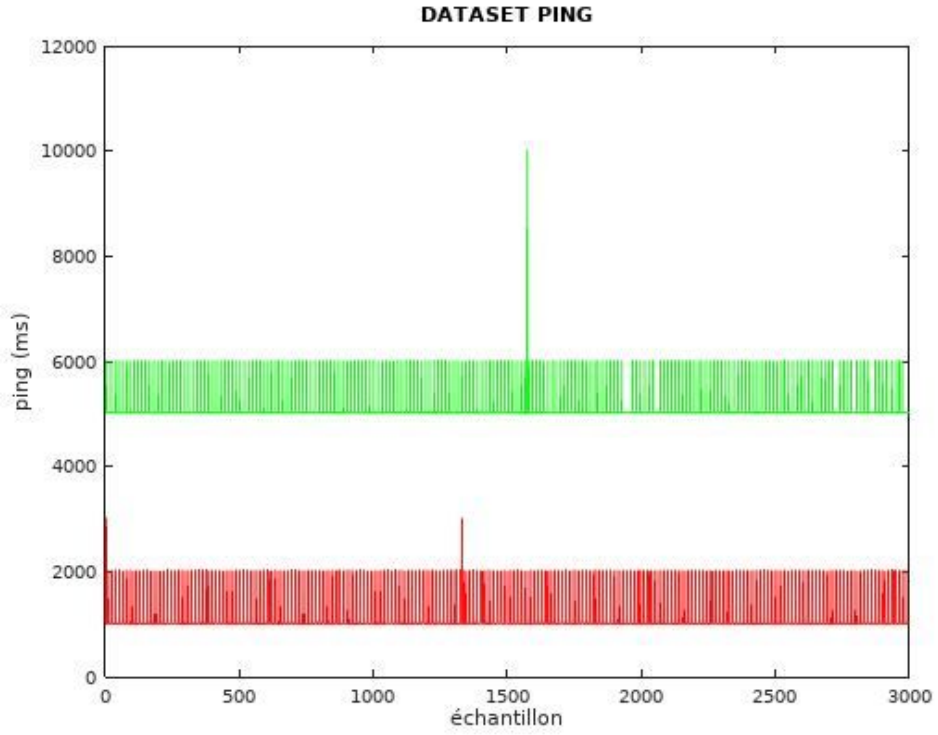
Classe (0, 1)	Puissance	Dist. euclidienne

L'utilisation de ces données sera expliquée dans la partie qui va suivre.

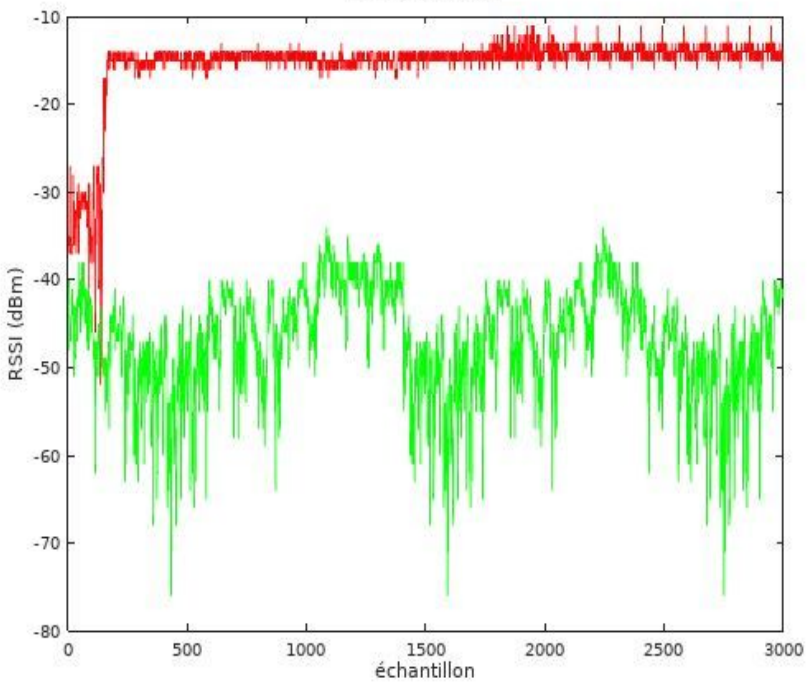
b) Architecture du réseau de neurones

Pour évaluer la performance du réseau de neurones et évaluer quelle architecture était nécessaire à la résolution de la problématique, nous avons testé empiriquement son comportement.

Ici, nous avons la chance d'avoir un dataset hétérogène. Par conséquent, La fonction à approximer ne nécessite pas une architecture profonde (comme du Deep Learning). Nous pouvons d'ailleurs directement le vérifier en affichant le dataset des RSSI et celui des PING:



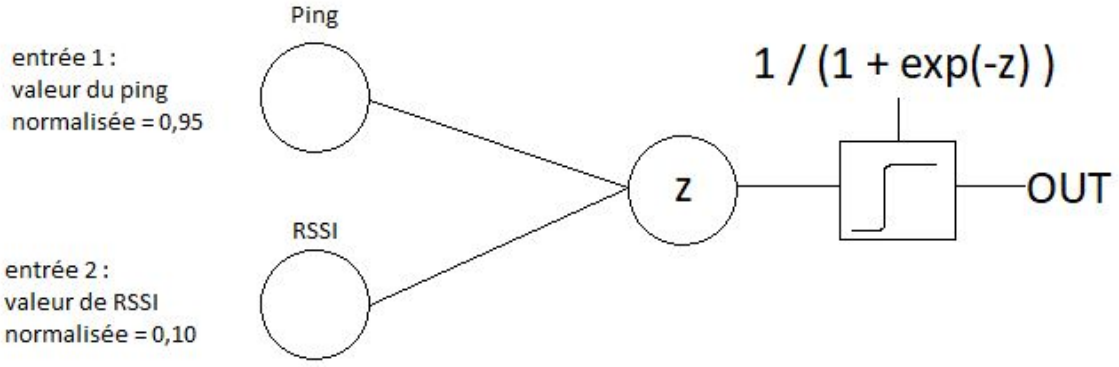
Répartition du ping avec trafic normal en vert et à risque en rouge



Répartition de la RSSI avec trafic normal en vert et à risque en rouge

Ainsi, après plusieurs tests, un réseau de neurones doté de deux neurones en entrée et d'un neurone de sortie répondait tout à fait à notre demande et convergeait très rapidement vers un précision importante (0,999) avec 50 epochs. En utilisant une fonction d'activation en échelon, nous pouvions obtenir une sortie avec uniquement des 1 et des 0. Pour un panel de valeurs plus large, il est préférable d'utiliser une fonction d'activation sigmoïdale comme dans l'exemple ci-dessous.

Prenons donc notre réseau de neurones :



Pour ces deux valeurs d'entrée, nous souhaitons obtenir 0 en sortie car c'est un comportement "normal" du trafic. Si nous prenons des poids équivalents au départ pour les 2 entrées : w1 = w2 = 0,5, calculons les nouveaux poids pour obtenir la sortie souhaitée : 0.

OUT à l'état initial :

$$OUT = \frac{1}{1+e^{-(0,95.0,5 + 0,10.0,5)}}$$

$$OUT = 0,628$$

Poids à la première itération :

$$w1 = (0 - 0,628) . 0,95 + 0,5 = -0,0966$$

$$w2 = (0 - 0,628) . 0,10 + 0,5 = 0,4372$$

Recalcul de la sortie OUT pour la première itération :

$$OUT = \frac{1}{1+e^{-(0,95.-0,0966 + 0,10.0,4372)}}$$

$$OUT = 0,487$$

Nous continuerons les itérations soit jusqu'à converger vers la valeur souhaitée, soit par nombre d'itérations maximal.

c) Enregistrement du modèle ANN

L'enregistrement des trainings est indispensable afin d'éviter le lancement d'un apprentissage lors de chaque exécution du programme. Ainsi, après appel de la méthode *fit*, il est possible d'enregistrer l'architecture du réseau de neurones mais aussi les valeurs de ses poids.

```
#################################################################################
#SAUVEGARDE DU TRAINING SET
#################################################################################
#Le modèle est mis au format JSON
classifier_json = classifier.to_json();
with open('C:/Users/Utilisateur/PFE/python/Training/training.json',"w") as json_file :
    json_file.write(classifier_json);
#Les poids sont mis en HDF5
classifier.save_weights("C:/Users/Utilisateur/PFE/python/Training/training.h5");
#################################################################################
```

Ensuite, le modèle est chargé dans un nouveau programme qui dispose donc de toutes les connaissances du réseau sans avoir effectué d'entrainement préliminaire.

6 - Classification

Au cours de ce projet, nous avons réalisé deux méthodes de classification. Une première utilisant un réseau de neurones artificiel et une seconde à partir de la méthode des K-plus proches voisins.

a) Réseau de neurones artificiel

Nous avons précédemment montré comment le réseau de neurones réalisait son apprentissage grâce aux données fournies. Il est ensuite capable, grâce à ses poids définis, de calculer quelle va être la sortie à partir des entrées PING et RSSI.
La sortie est ensuite arrondie à l'entier le plus près : 0 ou 1. À partir de cette valeur, il est possible de dire si oui ou non le modèle est une attaque.

b) K-plus proches voisins

La méthode des K-plus proches voisins consiste à prendre une valeur et à calculer sa distance euclidienne par rapport à celle du dataset. La matrice de connaissance et ensuite triée par dichotomie selon l'ordre croissant des distances. Si les K premières lignes appartiennent à une classe plus qu'une autre, alors nous associons le point étudié a cette classe.

$$d = \sqrt{(x1 - x2)^2}$$

x1 : valeur à classer
x2 : valeur d'une ligne du dataset

Pour cette classification, nous réalisons un calcul de distance pour le ping et un calcul de distance pour le rssi. Ainsi, l'utilisateur saura, via l'application web, s'il est attaqué en ping ou en puissance.

7 - Application web

L'application web est l'outil grâce auquel l'utilisateur est averti d'une éventuelle attaque. Celle-ci vient lire des fichiers csv contenant des informations sur le trafic et les affiche en temps réel.

À chaque trame reçue, nous enregistrons dans un fichier csv la date à laquelle celle-ci est apparue ainsi que :

- la température relevée
- la puissance reçue
- le ping
- la prédiction concernant la puissance
- la prédiction concernant le ping

Il y a donc 5 fichiers csv temporaires qui sont remis à 0 à chaque exécution du programme et qui s'incrémentent durant.

Ensuite, grâce à des outils fournis par la bibliothèque CanvasJS : https://canvasjs.com/, nous sommes parvenus à créer une interface graphique avec rafraichissement toutes les secondes.

Voici un aperçu de l'application web hébergée sur un serveur Apache2 en annexe 1.

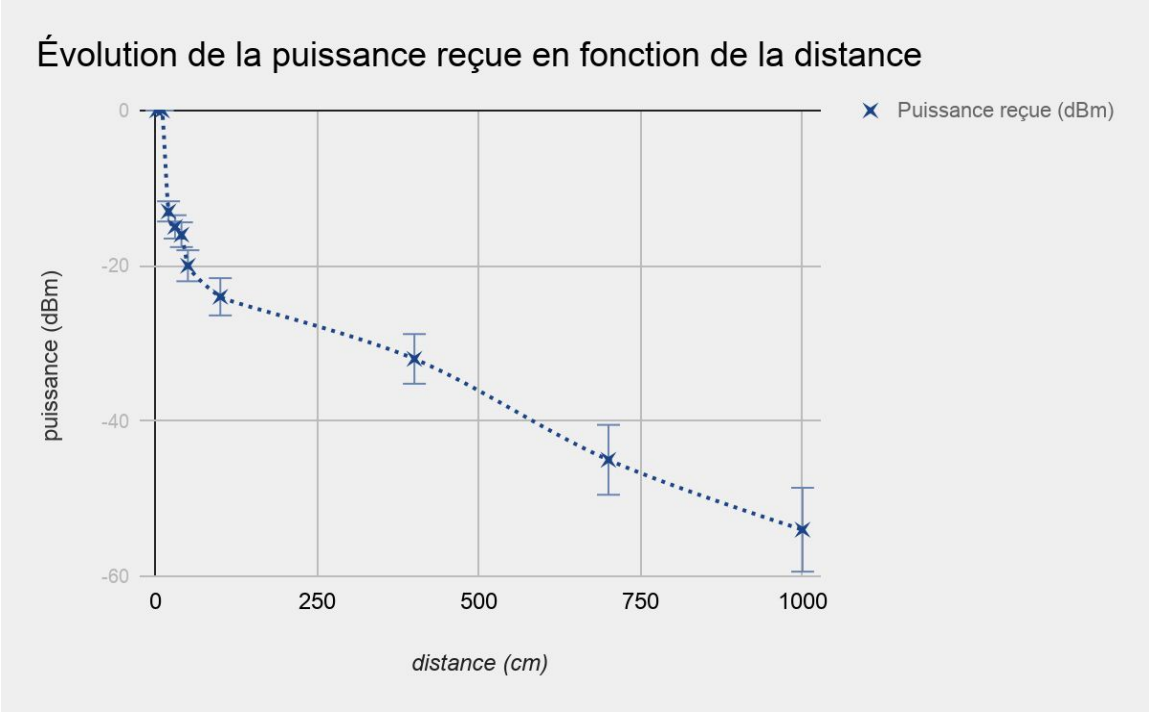
VI - Tests

1 - Variations des distances

Après avoir réalisé plusieurs batteries de tests, nous avons pu voir que la puissance était fortement volatile selon :

- la position de l'émetteur
- celle du récepteur
- la surface sur laquelle ils se trouvent
- leur orientation

À travers ce premier test, nous avons cherché à étudier le comportement de la puissance face à la distance séparant l'émetteur (14 dBm) et le récepteur.

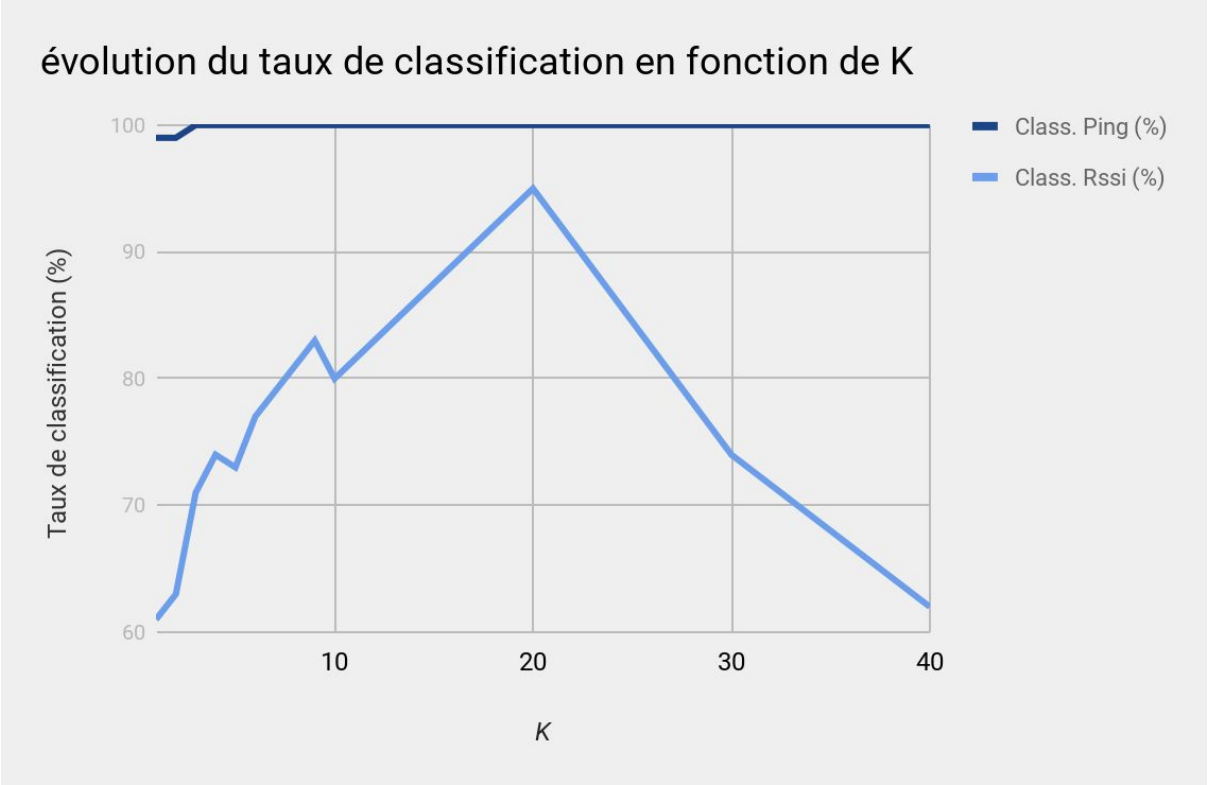


Sur des distances trop proches, aucun signal n'est reçu. C'était un premier facteur de classification : savoir que la puissance pouvait brouiller le signal.
Le tableau de valeur se trouve en annexe 2.

2 - Variations du k

Afin d'évaluer l'impacte que pouvait avoir la valeur de k dans la méthode des K-plus proches voisins, nous avons décidé de réaliser ce test en augmentant pas à pas sa valeur et en relevant le taux de succès lors des classifications.

Vous trouverez en annexe 3 le tableau de relevés correspondant aux détections d'attaque en puissance et en ping.



Ici, nous pouvons constater que la méthode des k plus proches voisins est très efficace pour le ping puisque les données des deux classes *ping_attack* et *ping_norm* sont fortement décorrélées.

Le dataset du RSSI est quant à lui plus homogène. La valeur de K a donc un fort impact sur les classifications. La frontière entre les deux classes et plus floue. Avec un K de 20, nous obtenons des résultats satisfaisants.

3 - Variations de vitesse d'émission

À 9600 bauds, un bit est traité environ toutes les 104 microsecondes. Les tests de vitesse d'émission ont été réalisés afin de voir dans quelles mesures le récepteur pouvait être mis hors service.

Nous avons constaté un DoS complet de la part du récepteur à partir d'un envoi toutes les 50 ms. À cette vitesse, nous n'observons plus aucune réponse de la part du récepteur. Une attaque par ping est tout de même détectée par notre système de classification par les K-plus proches voisins.

VII - Challenges rencontrés

Au cours de ce projet, différents challenges ont été rencontrés :

- définition des attaques
- portabilité du programme
- choix de la méthode de classification
- front-end

La définition des attaques sur le réseau LoRa a été l'élément du projet le plus complexe à définir. En effet, nous avons cherché à réaliser dans un premier temps une approche par ping et par rejet de trame tout comme il est possible de rencontrer ces phénomènes sur de l'IP.

Le code devait donc être adaptable à toutes sortes de réseaux. C'est le cas puisque les programme nécessitent simplement une modification des constantes concernant les trames. Tout est lu à partir du port série. Sur un autre réseau, il ne serait pas question de RSSI mais peut-être d'un autre facteur qui pourrait être, lui-aussi mis sous forme de matrice et contenir une connaissance supplémentaire pour le réseau de neurones ou encore pour la base de données des K-plus proches voisins.

Le choix de la méthode de classification dépendait des tests réalisés. Dans un premier temps, nous avons conçu un algorithme autour d'un réseau de neurones. Les prédictions étaient précises à environ 80%, un résultat convenable sur autant de données. Cependant, le traitement s'avérait être trop lent pour faire le lien avec une application web affichant les données en temps réel.

L'application a donc été orientée vers l'utilisation d'une classification utilisant une intelligence artificielle probabiliste : Les K-plus proches voisins.
Cette méthode nécessite un tri par dichotomie avec un coup de traitement correspondant à la taille du vecteur à trier.

Jusqu'à présent, le javascript nous était inconnu. À travers la création d'une interface graphique pour peupler l'application Web, nous avons pu utiliser une bibliothèque : canvasJS, dans le but de concevoir nos graphiques interactifs. Le challenge, pour cette partie était de récupérer les données présentes dans les csv et de les manipuler.

VIII - Conclusion

L'apprentissage automatique pour la détection d'attaques par déni de services est un projet très actuel utilisant des technologies prometteuses. Il met en relation le réseau et le traitement des données relatives à l'utilisation d'une intelligence artificielle.

L'objectif, était de travailler sur la création d'un dataset correspondant à un trafic normal et un trafic menaçant, pour ensuite exploiter les données dans un algorithme de prédiction.

Pour générer des trames, nous avons utilisé un émetteur et un récepteur LoRa réglés sur le même canal et la même fréquence. L'émetteur envoi une trame composée d'un caractère de début et de fin, d'une entête, d'une température aléatoire et d'un compteur de trame. Dans le cadre d'un trafic normal, une trame est générée toutes les 5s et toutes les 1s pour les attaques.

Nous avons choisi de distinguer le comportement du trafic en deux classes :

- 0 : trafic normal
- 1 : trafic à risque

Ces deux classes sont distinguées par le ping et par la puissance des messages reçus.

Deux méthodes de classification ont été testées :

- méthode bio-inspirée : réseau de neurones artificiel
- méthode probabiliste : K-plus proches voisins

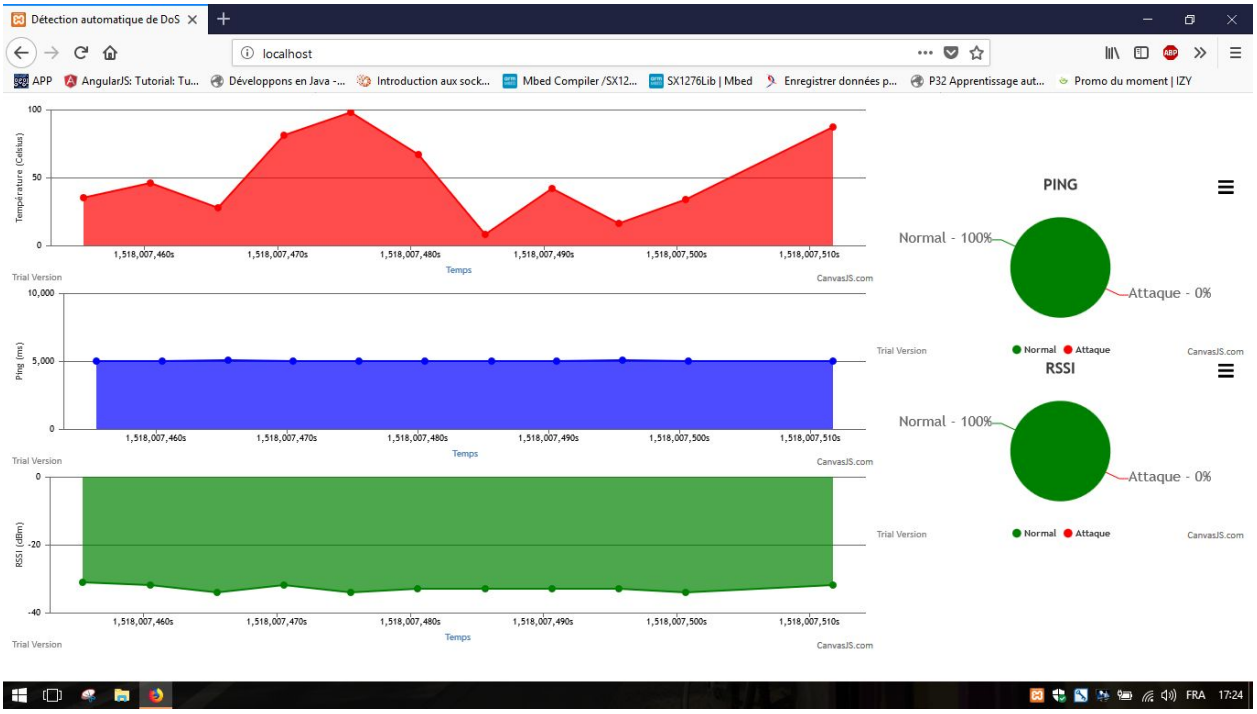
Le réseau de neurones est une véritable boite noire où les performances ont été testées empiriquement. Plusieurs architectures ont donné forme à des résultats significatifs. Nous avons retenu un réseau avec deux neurones en entrée et un seul en sortie activée par une fonction sigmoïdale pour obtenir plus de précision dans les résultats.

Cependant, cette méthode s'est avérée lente lors des tentatives de prédiction en temps réel. Pour poursuivre le projet, une méthode des K-plus proches voisins a été préférée. Une variation de K a été effectuée pour obtenir de meilleurs résultats. Avec cette méthode, les prédictions s'avèrent précises pour un K de 20. La base de données est repeuplée à chaque prédiction. Il faudra veiller à ce que le temps de calcul visant à trier les matrices ne dépasse un certain seuil. Auquel cas, tout comme le réseau de neurones, nous verrions apparaître un retard au niveau des prédictions.

Avec un algorithme plus simple à implémenter et également plus facilement contrôlable, nous sommes parvenus à de meilleurs résultats concernant l'apprentissage pour la détection de DoS.

Ce projet serait adapté à l'écoute d'un trafic sur WireShark. Les trames pourraient être étudiées quasiment en temps réel et prévenir les équipes d'une éventuelle attaques par mail - SMS ou autre moyen de communication.

Annexes



annexe 1 - Aperçu de l'application web

Distance (cm)	Puissance moyenne reçue (dBm)
2	0
10	0
20	-13
30	-15
40	-16
50	-20
100	-24
400	-32
700	-45
1000	-54

annexe 2 - Tableau d'essais de puissance en fonction de la distance séparant l'émetteur et le récepteur

K	Taux de bonnes classifications PING (%)	Taux de bonnes classifications RSSI (%)
1	99	61
2	99	63
3	100	71
4	100	74
5	100	73
6	100	77
7	100	79
8	100	81
9	100	83
10	100	80
20	100	95
30	100	74
40	100	62

annexe 3 - Taux de classification en fonction de la valeur de k

Bibliographie

Livres :

Artificial intelligence: a guide to intelligent systems M Negnevitsky - 2005

Artificial intelligence: a modern approach (International Edition)* SJ Russell, P Norvig - 2002

INTELLIGENCE ARTIFICIELLE PROBABILISTE - RECONNAISSANCE DE FORMES - APPRENTISSAGE MACHINE, Professeur Jean Rouat - 2017

Liens :

https://www.udemy.com/deeplearning/learn/v4/content

http://www.journaldunet.com/ebusiness/internet-mobile/1197635-lora-reseau-differences-sigfox/

https://os.mbed.com/users/GregCr/code/SX1276Lib/docs/tip/classRadio.html

https://www.mbed.com/en/

http://www.st.com/en/microcontrollers/stm32-32-bit-arm-cortex-mcus.html

https://openclassrooms.com/courses/initiez-vous-au-machine-learning/tp-entrainez-le-modele-des-k-plus-proches-voisins-k-nn

https://keras.io/

http://www.numpy.org/

https://pandas.pydata.org/

https://canvasjs.com/

https://www.apachefriends.org/fr/index.html

https://www.zingchart.com/docs/tutorials/loading-data/parsing-csv-files/

https://www.youtube.com/watch?v=trWrEWfhTVg