INPT
المعهد الوطني للبريد والمواصلات
ⵓⵙⴷⵖ ⵏⵓⵟⵏⴰⵢ ⵏ ⵜⵙⵓⵡⵉⵔⵉⵏ ⴷ ⵜⵙⵡⵉⵍⵜ
Institut National des Postes et Télécommunications

anrt
agence nationale de réglementation des télécommunications
الوكالة الوطنية لتقنين المواصلات

Compte Rendu Projet :

WSN-Réseau de capteurs sans fil

Travail realisé par : ***Yassine lazrak***

Filière : ***SUD2***

Prérequis

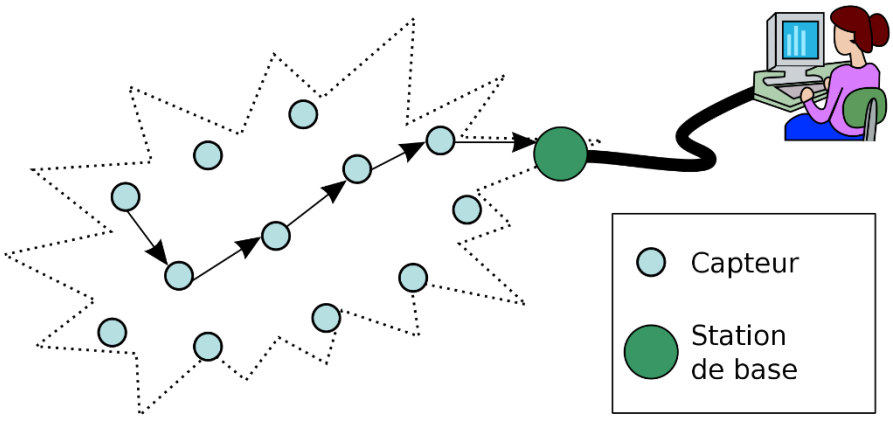
A la fois facile à apprendre et supporté par une large communauté, Python est particulièrement pratique dans le cadre du développement IoT. Sa syntaxe est claire et simple, et attire de plus en plus de développeurs. Python est le langage idéal pour les microcontrôleurs les plus populaires du marché, Raspberry Pi .

Tkinter est la bibliothèque graphique OpenSource pour le langage Python, permettant la création d'interfaces graphiques. Elle vient d'une adaptation de la bibliothèque graphique Tk écrite pour Tcl .

Matplotlib est une bibliothèque du langage de programmation Python destinée à tracer et visualiser des données sous forme de graphiques.

Protocole Leach

LEACH : La hiérarchie de clustering adaptatif à faible énergie est un protocole MAC basé qui est intégré au clustering et à un protocole de routage simple dans les réseaux de capteurs sans fil (WSN). L'objectif de LEACH est de réduire la consommation d'énergie nécessaire pour créer et maintenir des clusters afin d'améliorer la durée de vie d'un réseau de capteurs sans fil.



Problématique

On développera une Application en Python avec une interface graphique pour le problème de clustering LEACH en utilisant la formule suivante :

$$T(n)_{new} = \frac{P}{1 - P\left(r \bmod \frac{1}{P}\right)} \frac{E_{n\_current}}{E_{n\_max}}$$

Tel que :

- P : c'est le pourcentage des clusterHeads .
- R : c'est le nombre de Rounds .
- En_Current : c'est l'énergie disponible dans un nœud .
- En_max : c'est l'énergie maximale que peut avoir un nœud .

On notera que :

- En_Current : est aléatoire pour chaque nœud .
- n est le numéro qui caractérise le nœud .

Implémentation

Tout d'abord on fixe les coordonnés de notre champ (ou les nœuds seront disposés). Ainsi que le positionnement de notre Base Station et l'énergie initiale .

```
class Model:
    def __init__(self, n,p,Eo):
        self.n = n
        self.x = 1000
        self.y = 1000
        self.sink_x = self.x * 0.5
        self.sink_y = self.y * 0.5
        self.sinkE = 100
        self.p: float = p
        self.Eo: float = Eo
        self.Eelec: float = 50 * 0.000000001
        self.ETX: float = 50 * 0.000000001
        self.ERX: float = 50 * 0.000000001
        self.Efs: float = 10e-12
        self.Emp: float = 0.0013 * 0.000000000001
        self.EDA: float = 5 * 0.000000001
        self.do: float = sqrt(self.Efs / self.Emp)
        self.rmax = 200
        self.data_packet_len = 4000
        self.hello_packet_len = 100
        self.NumPacket = 10
        self.RR: float = 0.5 * self.x * sqrt(2)
```

Et on procède à la création de nos nœuds avec cette fonction :

```
def create_sensors(my_model: Model):
    n = my_model.n
    sensors = [Sensor() for _ in range(n + 1)]

    sensors[n].xd = my_model.sink_x
    sensors[n].yd = my_model.sink_y
    sensors[n].E = my_model.sinkE
    sensors[n].id = my_model.n
    sensors[n].type = 'S'

    for i, sensor in enumerate(sensors[:-1]):

        sensor.xd = random.randint(1, my_model.x)
        sensor.yd = random.randint(1, my_model.y)

        sensor.G = 0
        sensor.df = 0
        sensor.type = 'N'
        sensor.E = my_model.Eo
        sensor.id = i
        sensor.RR = my_model.RR
        sensor.MCH = n
        sensor.dis2sink = sqrt(pow((sensor.xd - sensors[-1].xd), 2) + pow((sensor.yd - sensors[-1].yd), 2))


    return sensors
```

On appellera cette fonction à chaque début de round :

```python
def reset(Sensors: list[Sensor], my_model: Model, round_number):
    for sensor in Sensors[:-1]:


        AroundClear = 1 / my_model.p
        if round_number % AroundClear == 0:
            sensor.G = 0

        sensor.MCH = my_model.n

        if sensor.type != 'S':
            sensor.type = 'N'
        sensor.dis2ch = inf

    srp = 0
    sdp = 0
    rdp = 0

    return srp, rrp, sdp, rdp
```

On implémente notre fonction T :

```python
def start(sensors: list[Sensor], my_model, round_number: int, state: int):
    CH = []
    n = my_model.n

    for sensor in sensors[:-1]:

        if sensor.E > 0 and sensor.G <= 0:
            a=sensor.E/sensor.Eo
            b=round(sensor.rs/(1/my_model.p))
            c=a+b*(1-a)

            temp_rand = random.uniform(0, 1)


            if(state==2):
                value = (my_model.p / (1 - my_model.p * (round_number % (1 / my_model.p))))*a


            if temp_rand <= value:
                CH.append(sensor.id)
                sensor.type = 'C'
                sensor.rs=0
                sensor.G = round(1 / my_model.p) - 1
            else:
                sensor.rs=sensor.rs+1

    return CH
```

Application

On initialise notre interface graphique :

```
#Mise en Place de l'interface graphique :

root = tk.Tk()
root.geometry('1300x800')
root.title("Projet WSN")
right_frame = tk.Frame(root, bg='#121111', bd=1)
right_frame.place(relx=0.3, rely=0.05, relwidth=0.65, relheight=0.9)
left_frame = tk.Frame(root, bg='#4C4E52')
left_frame.place(relx=0.03, rely=0.05, relwidth=0.30, relheight=0.9)

#Gestion du Placement du Graphe :

figure = plt.Figure(figsize=(4, 6),dpi=80)
figure.set_size_inches(10.5, 5.5)
ay = figure.add_subplot(111)
line = FigureCanvasTkAgg(figure, right_frame)
line.get_tk_widget().pack(side=tk.LEFT, fill=tk.BOTH,expand=1)
```

Voici notre fonction qui regroupe le Back end avec le front end:

```
def Clustering(state:int):
    global App, n, my_model, alive_sensors,sum_energy_left_all_nodes
    n=int(entry1.get())
    p=float(entry2.get())
    En_max=float(entry3.get())
    roundReseau=int(entry4.get())
    App=Run.Simulation(n,p,En_max,roundReseau,state)
    n,my_model,alive_sensors,sum_energy_left_all_nodes,noeuds,model,tour=App.start()
    ay.clear()
    capture(noeuds,model,tour)
    line.draw()
```

Et voici a quoi ressemble le résultat après exécution :