PRESENTATION DE AKRAM-SIM

FAIT PAR :

KIES AKRAM-WALID

LORASIM ?

Le simulateur le plus populaire pour LoRaWAN

Un simulateur d'évènements discrets basé sur python avec la bib de simulation SimPy.

Tous les autres simulateurs pour LoRaWAN sont basés sur LoRaSim, ces derniers étendent LoRaSim pour plusieurs applications IoT (Internet of things).

AVANTAGES DE LORASIM

- C'est la base de tous les autres simulateurs.

INCONVENIENTS DE LORASIM

- Il ne prend pas en considération les acquittements (ACK).
- Les paramètres (SF, BW, CR...) ne sont pas paramétrés.
- Les calculs (énergie...) ne sont pas précis.
- Il ne prend pas en considération la taille des paquets.
- Les paramètres sont relatifs à chaque noeud.

LORA FREE ?

C'EST L'UN DES SIMULATEURS DE LORAWAN BASÉ SUR LORASIM.

LE CHOIX DE FREE A ÉTÉ FAIT CAR C'EST LE SIMULATEUR LE PLUS APPROPRIÉ SELON LES STATISTIQUES (VOIR TABLEAU COMPARATIF DES SIMULATEURS OPEN SOURCE).

AVANTAGES DE FREE

- Il prend en considération les aquitements (ACK)
- Les calculs (énergie...) sont bien précis.
- Il prend en considération la taille des paquets.
- Les paramètres sont relatifs à chaque paquet.

INCONVÉNIENTS DE FREE

LES PARAMÈTRES (SF, BW, CR...) NE SONT PAS PARAMÉTRÉS.

IL NE PREND PAS EN CONSIDÉRATION LE CAS DE MULTIPLES STATIONS DE BASE.

SIMULATEUR AKRAM-SIM

En se basant sur les avantages et les inconvénients des deux simulateurs étudiés (cités précédemment) et en faisant plusieurs simulations

J'ai eu l'idée de réaliser le nouveau simulateur (appelé AKRAM-SIM)

En lisant attentivement le code des deux simulateurs et en se basant sur ces derniers.

POINTS ESSENTIELS DE AKRAM-SIM

Il prend en considération les acquittements (ACK), les collisions.

Les paramètres (SF, BW, CR...) sont paramétrés.

Les calculs (énergie...) sont plus précis (en tenant compte tous les paramètres).

Les paramètres sont relatifs à chaque paqutes.

Il prend en considération le cas de multiples stations de base.

Deux fichiers exécutables (OneBS et MulBS).

FICHIER ONEBS

- C'est un exécutable pour la simulation avec une seule station de base.
- **Usage :**

./OneBS.py <nodes> <avgsend> <experiment> <simtime> <datasize> <SF> <BW> [collision]

D'où :

Nodes : le nombre de nœud

Avgsend : le temps moyen de transsmision.

Experiment :

FICHIER ONEBS

experiment = 0 : utiliser les paramètres avec le débit de données le plus lent (SF12, BW125, CR4 / 8).

experiment = 1 : similaire à l'expérience 0, mais utilisez un choix aléatoire de 3 fréquences de transmission.

experiment = 2 : utiliser les paramètres avec le débit de données le plus rapide (SF6, BW500, CR4 / 5).

experiment = 3 : optimiser le paramètre par nœud en fonction de la distance à la passerelle.

experiment = 4 : utiliser les paramètres définis par défaut dans LoRaWAN (SF12, BW125, CR4 / 5).

experiment = 5 : similaire à l'expérience 3, mais optimise également la puissance de transmission.

experiment >= 6 : utiliser les paramètres <SF> <BW>

FICHIER ONEBS

Simtime : durée totale de la simulation en millisecondes.

Datasize : taille du paquet à transmettre.

SF : facteur d'étalement [7-12].

BW : bande passante [125,250,500].

Collision : 0 pour simple collision

1 pour full collision.

CLASS MYNODE

```
class myNode():
    def __init__(self, nodeid, bs, period, packetlen):
        self.nodeid = nodeid
        self.period = period
        self.bs = bs
        self.x = 0
        self.y = 0
        self.sent = 0
        self.coll = 0
        self.noack = 0
        self.acklost = 0
        self.recv = 0
        self.lost = 0
        self.rxtime = 0
        self.lstretans = 0
        # this is very complex prodecure for placing nodes
        # and ensure minimum distance between each pair of nodes
        found = 0
        rounds = 0
        global nodes
        while (found == 0 and rounds < 100):
            a = random.random()
            b = random.random()
            if b<a:
                a,b = b,a
            posx = b*maxDist*math.cos(2*math.pi*a/b)+bsx
            posy = b*maxDist*math.sin(2*math.pi*a/b)+bsy
            if len(nodes) > 0:
                for index, n in enumerate(nodes):
                    dist = np.sqrt(((abs(n.x-posx))**2)+((abs(n.y-posy
                    if dist >= 10:
                        found = 1
                        self.x = posx
                        self.y = posy
                    else:
                        rounds = rounds + 1
                        if rounds == 100:
                            print "could not place new node, giving up
                            exit(-1)
            else:
                print "first node"
                self.x = posx
                self.y = posy
                found = 1
        self.dist = np.sqrt((self.x-bsx)*(self.x-bsx)+(self.y-bsy)*(se
        print('node %d' %nodeid, "x", self.x, "y", self.y, "dist: ", s

        self.packet = myPacket(self.nodeid, packetlen, self.dist)
        self.sent = 0
```

CLASS MYPACKET

```
class myPacket():
    def __init__(self, nodeid, plen, distance):
        global experiment
        global Ptx
        global gamma
        global d0
        global var
        global Lpld0
        global GL

        self.nodeid = nodeid
        self.txpow = Ptx

        # randomize configuration values
        self.sf = random.randint(6,12)
        self.cr = random.randint(1,4)
        self.bw = random.choice([125, 250, 500])

        # for certain experiments override these
        if experiment==1 or experiment == 0:
            self.sf = 12
            self.cr = 4
            self.bw = 125

        # for certain experiments override these
        if experiment==2:
            self.sf = 6
            self.cr = 1
            self.bw = 500
        # lorawan
        if experiment == 4:
            self.sf = 12
            self.cr = 1
            self.bw = 125
        if experiment > 5:
            self.sf = SF
            self.cr = 1
            self.bw = BW


        # for experiment 3 find the best setting
        # OBS, some hardcoded values
        Prx = self.txpow  ## zero path loss by default

        # log-shadow
        Lpl = Lpld0 + 10*gamma*math.log10(distance/d0)
        print "Lpl:", Lpl
        Prx = self.txpow - GL - Lpl

        if (experiment == 3) or (experiment == 5):
            minairtime = 9999
            minsf = 0
```

CLASS MYPACKET

```
    if (experiment == 3) or (experiment == 5):
        minairtime = 9999
        minsf = 0
        minbw = 0

        print "Prx:", Prx

        for i in range(0,6):
            for j in range(1,4):
                if (sensi[i,j] < Prx):
                    self.sf = int(sensi[i,0])
                    if j==1:
                        self.bw = 125
                    elif j==2:
                        self.bw = 250
                    else:
                        self.bw=500
                    at = airtime(self.sf, 1, plen, self.bw)
                    if at < minairtime:
                        minairtime = at
                        minsf = self.sf
                        minbw = self.bw
                        minsensi = sensi[i, j]
        if (minairtime == 9999):
            print "does not reach base station"
            exit(-1)
        print "best sf:", minsf, " best bw: ", minbw, "best airtime:", minairtime
        self.rectime = minairtime
        self.sf = minsf
        self.bw = minbw
        self.cr = 1

        if experiment == 5:
            # reduce the txpower if there's room left
            self.txpow = max(2, self.txpow - math.floor(Prx - minsensi))
            Prx = self.txpow - GL - Lpl
            print 'minsesi {} best txpow {}'.format(minsensi, self.txpow)

    # transmission range, needs update XXX
    self.transRange = 150
    self.pl = plen
    self.symTime = (2.0**self.sf)/self.bw
    self.arriveTime = 0
    self.rssi = Prx
    # frequencies: lower bound + number of 61 Hz steps
    self.freq = 860000000 + random.randint(0,2622950)

    # for certain experiments override these and
    # choose some random frequences
    if experiment == 1:
        self.freq = random.choice([860000000, 864000000, 868000000])
    else:
```

CLASS MYPACKET

```
else:
    self.freq = 860000000

print "frequency" ,self.freq, "symTime ", self.symTime
print "bw", self.bw, "sf", self.sf, "cr", self.cr, "rssi", self.rssi
self.rectime = airtime(self.sf,self.cr,self.pl,self.bw)
print "rectime node ", self.nodeid, "  ", self.rectime
# denote if packet is collided
self.collided = 0
self.processed = 0
self.lost = False
self.perror = False
self.acked = 0
self.acklost = 0
```

CHECK COLLISIONS

```
# check for collisions at base station
# Note: called before a packet (or rather node) is inserted into the list
def checkcollision(packet):
    col = 0 # flag needed since there might be several collisions for packet
    processing = 0
    for i in range(0,len(packetsAtBS)):
        if packetsAtBS[i].packet.processed == 1:
            processing = processing + 1
    if (processing > maxBSReceives):
        print "too long:", len(packetsAtBS)
        packet.processed = 0
    else:
        packet.processed = 1

    if packetsAtBS:
        print "CHECK node {} (sf:{} bw:{} freq:{:.6e}) others: {}".format(
             packet.nodeid, packet.sf, packet.bw, packet.freq,
             len(packetsAtBS))
        for other in packetsAtBS:
            if other.nodeid != packet.nodeid:
               print ">> node {} (sf:{} bw:{} freq:{:.6e})".format(
                    other.nodeid, other.packet.sf, other.packet.bw, other.packet.freq)
               # simple collision
               if frequencyCollision(packet, other.packet) \
                   and sfCollision(packet, other.packet):
                   if full_collision:
                       if timingCollision(packet, other.packet):
                           # check who collides in the power domain
                           c = powerCollision(packet, other.packet)
                           # mark all the collided packets
                           # either this one, the other one, or both
                           for p in c:
                               p.collided = 1
                               if p == packet:
                                   col = 1
                       else:
                           # no timing collision, all fine
                           pass
                   else:
                       packet.collided = 1
                       other.packet.collided = 1  # other also got lost, if it wasn't lost already
                       col = 1
        return col
    return 0
```

FREQUENCY COLLISION

```
def frequencyCollision(p1,p2):
    if (abs(p1.freq-p2.freq)<=120 and (p1.bw==500 or p2.freq==500)):
        print "frequency coll 500"
        return True
    elif (abs(p1.freq-p2.freq)<=60 and (p1.bw==250 or p2.freq==250)):
        print "frequency coll 250"
        return True
    else:
        if (abs(p1.freq-p2.freq)<=30):
            print "frequency coll 125"
            return True
        #else:
    print "no frequency coll"
    return False
```

SF COLLISION

```
def sfCollision(p1, p2):
    if p1.sf == p2.sf:
        print "collision sf node {} and node {}".format(p1.nodeid, p2.nodeid)
        # p2 may have been lost too, will be marked by other checks
        return True
    print "no sf collision"
    return False
```

POWER COLLISION

```
def powerCollision(p1, p2):
    powerThreshold = 6 # dB
    print "pwr: node {0.nodeid} {0.rssi:3.2f} dBm node {1.nodeid} {1.rssi:3.2f} dBm; diff {2:3.2f} dBm".format(p1, p2, round(p1.rssi - p2.rssi,2))
    if abs(p1.rssi - p2.rssi) < powerThreshold:
        print "collision pwr both node {} and node {}".format(p1.nodeid, p2.nodeid)
        # packets are too close to each other, both collide
        # return both packets as casualties
        return (p1, p2)
    elif p1.rssi - p2.rssi < powerThreshold:
        # p2 overpowered p1, return p1 as casualty
        print "collision pwr node {} overpowered node {}".format(p2.nodeid, p1.nodeid)
        return (p1,)
    print "p1 wins, p2 lost"
    # p2 was the weaker packet, return it as a casualty
    return (p2,)
```

TIMING COLLISION

```
def timingCollision(p1, p2):
    # assuming p1 is the freshly arrived packet and this is the last check
    # we've already determined that p1 is a weak packet, so the only
    # way we can win is by being late enough (only the first n - 5 preamble symbols overlap)

    # assuming 8 preamble symbols
    Npream = 8

    # we can lose at most (Npream - 5) * Tsym of our preamble
    Tpreamb = 2**p1.sf/(1.0*p1.bw) * (Npream - 5)

    # check whether p2 ends in p1's critical section
    p2_end = p2.addTime + p2.rectime
    p1_cs = env.now + Tpreamb
    print "collision timing node {} ({},{},{}) node {} ({},{})".format(
        p1.nodeid, env.now - env.now, p1_cs - env.now, p1.rectime,
        p2.nodeid, p2.addTime - env.now, p2_end - env.now
    )
    if p1_cs < p2_end:
        # p1 collided with p2 and lost
        print "not late enough"
        return True
    print "saved by the preamble"
    return False
```

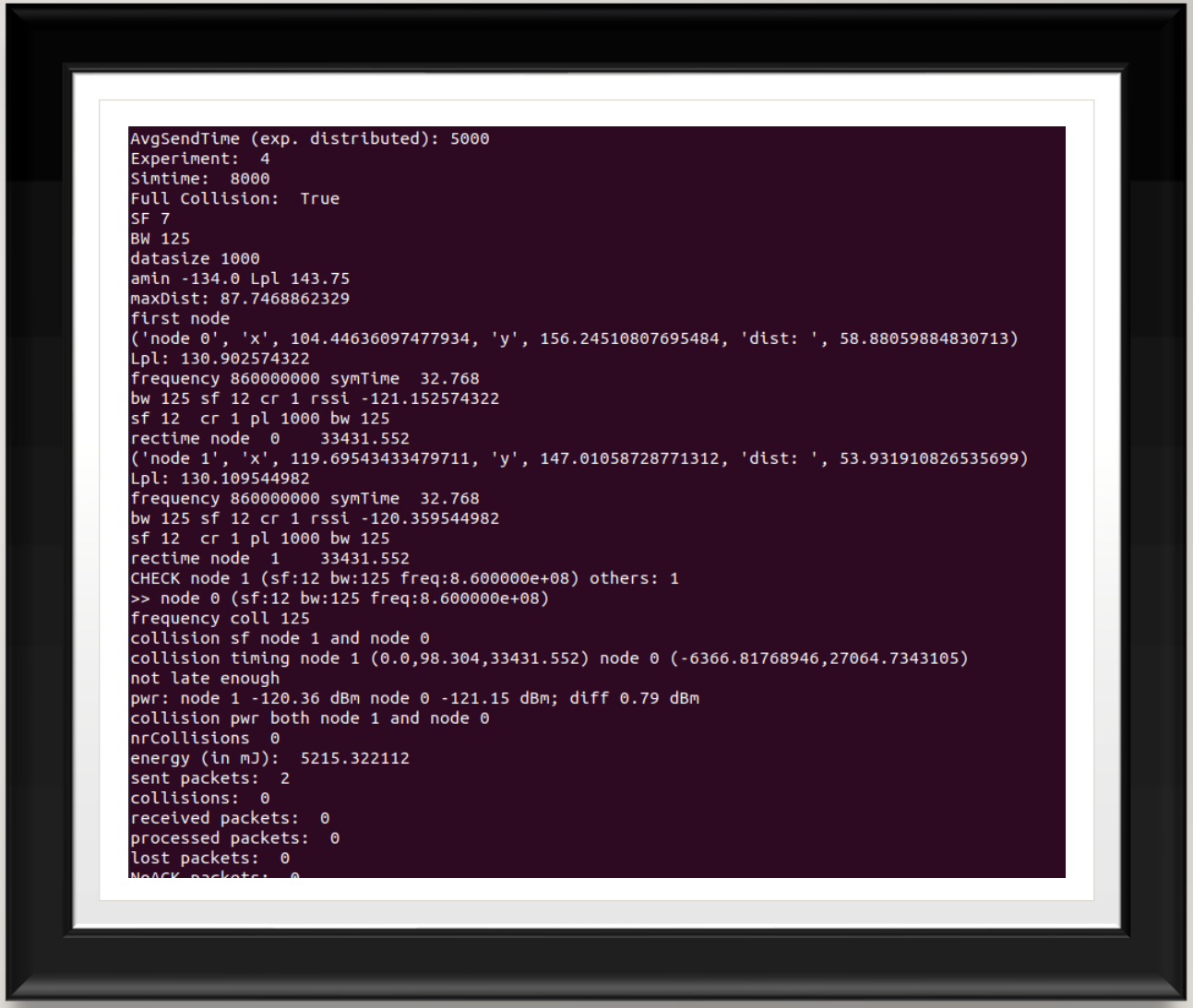
CHECK ACK

```
def checkACK(packet):
    global  nearstACK1p
    global  nearstACK10p
    # check ack in the first window
    chanlindex=[872000000, 864000000, 860000000].index(packet.freq)
    timeofacking = env.now + 1  # one sec after receiving the packet
    if (timeofacking >= nearstACK1p[chanlindex]):
        # this packet can be acked
        packet.acked = 1
        tempairtime = airtime(packet.sf, CodingRate, AckMessLen+LorawanHeader, Bandwidth)
        nearstACK1p[chanlindex] = timeofacking+(tempairtime/0.01)
        nodes[packet.nodeid].rxtime += tempairtime
        return packet.acked
    else:
        # this packet can not be acked
        packet.acked = 0
        Tsym = (2**packet.sf)/(Bandwidth*1000) # sec
        Tpream = (8 + 4.25)*Tsym
        nodes[packet.nodeid].rxtime += Tpream

    # chcek ack in the second window
    timeofacking = env.now + 2  # two secs after receiving the packet
    if (timeofacking >= nearstACK10p):
        # this packet can be acked
        packet.acked = 1
        tempairtime = airtime(12, CodingRate, AckMessLen+LorawanHeader, Bandwidth)
        nearstACK10p = timeofacking+(tempairtime/0.1)
        nodes[packet.nodeid].rxtime += tempairtime
        return packet.acked
    else:
        # this packet can not be acked
        packet.acked = 0
        Tsym = (2.0**12)/(Bandwidth*1000.0) # sec
        Tpream = (8 + 4.25)*Tsym
        nodes[packet.nodeid].rxtime += Tpream
        return packet.acked
#
```

TRANSMIT

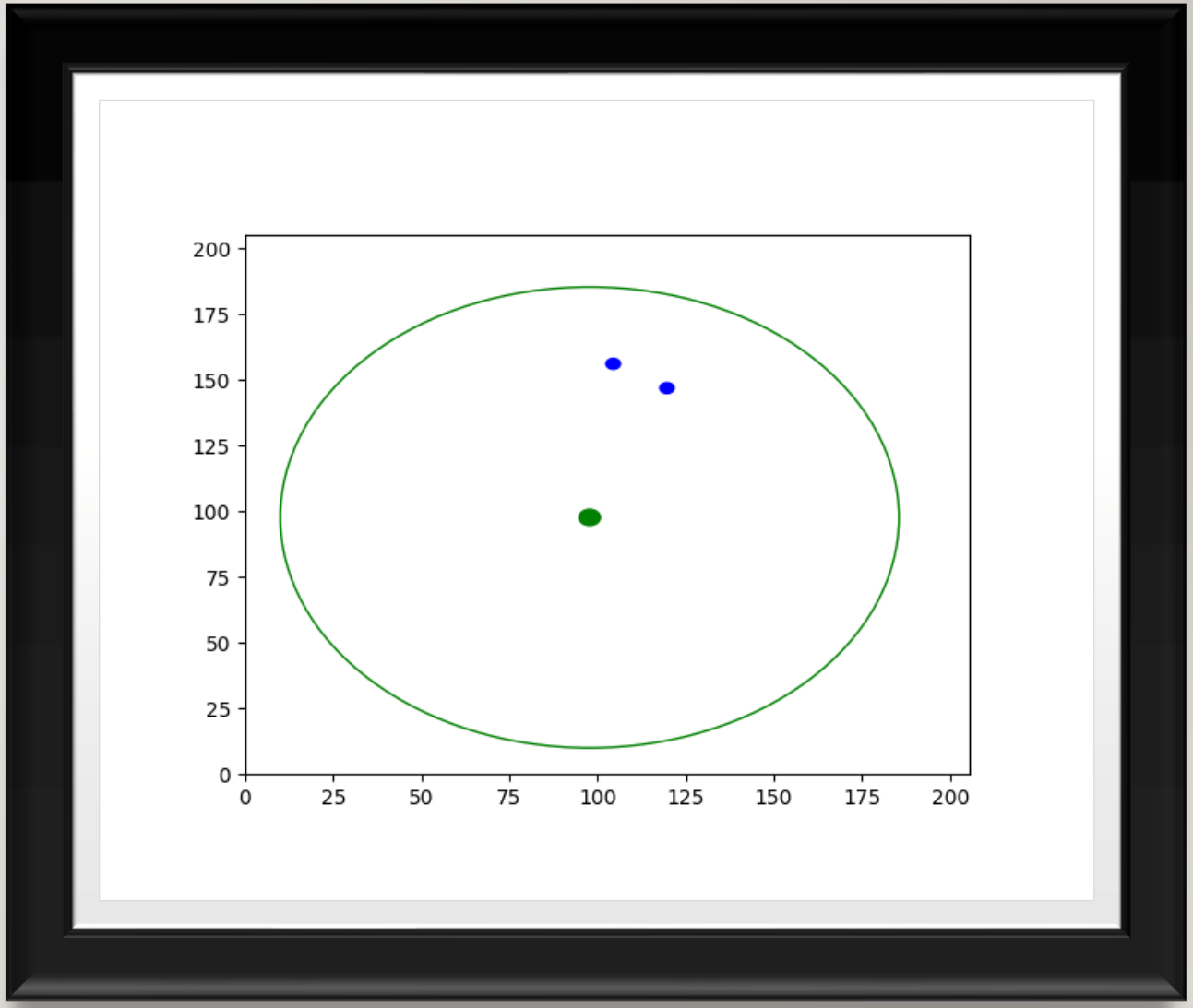
```
def transmit(env,node):
    while True:
        yield env.timeout(random.expovariate(1.0/float(node.period)))

        # time sending and receiving
        # packet arrives -> add to base station

        node.sent = node.sent + 1

        global packetSeq
        packetSeq = packetSeq + 1

        global nrBS
        for bs in range(0, nrBS):
            if (node in packetsAtBS):
                print "ERROR: packet already in"
            else:
                sensitivity = sensi[node.packet[bs].sf - 7, [125,250,500].index(node.packet[bs].bw) + 1]
            if node.packet[bs].rssi < sensitivity:
                print "node {}: packet will be lost".format(node.nodeid)
                node.packet[bs].lost = True
            else:
                node.packet[bs].lost = False
                if (per(node.packet[bs].sf,node.packet[bs].bw,node.packet[bs].cr,node.packet[bs].rssi,node.packet[bs].pl) < random.uniform(0,1)):
                    # OK CRC
                    node.packet[bs].perror = False
                else:
                    # Bad CRC
                    node.packet[bs].perror = True
                # adding packet if no collision

                if (checkcollision(node.packet[bs])==1):
                    node.packet[bs].collided = 1

                else:
                    node.packet[bs].collided = 0
            yield env.timeout(node.packet[0].rectime)

            if (node.packet[bs].lost == 0\
                and node.packet[bs].perror == False\
                and node.packet[bs].collided == False\
                and checkACK(node.packet[bs])):
                node.packet[bs].acked = 1
            # the packet can be acked
            # check if the ack is lost or not
                if((14 - Lpld0 - 10*gamma*math.log10(node.distances[bs]/d0) - np.random.normal(-var, var)) > sensi[node.packet[bs].sf-7, [125,250,500].index(node.packet[bs].bw) + 1]):
                # the ack is not lost
                    node.packet[bs].acklost = 0
                else:
            # ack is lost
                    node.packet[bs].acklost = 1
            else:
                node.packet[bs].acked = 0
```

EXÉCUTION DU FICHIER ONEBS



```
AvgSendTime (exp. distributed): 5000
Experiment:  4
Simtime:  8000
Full Collision:  True
SF 7
BW 125
datasize 1000
amin -134.0 Lpl 143.75
maxDist: 87.7468862329
first node
('node 0', 'x', 104.44636097477934, 'y', 156.24510807695484, 'dist: ', 58.88059884830713)
Lpl: 130.902574322
frequency 860000000 symTime  32.768
bw 125 sf 12 cr 1 rssi -121.152574322
sf 12  cr 1 pl 1000 bw 125
rectime node  0    33431.552
('node 1', 'x', 119.69543433479711, 'y', 147.01058728771312, 'dist: ', 53.931910826535699)
Lpl: 130.109544982
frequency 860000000 symTime  32.768
bw 125 sf 12 cr 1 rssi -120.359544982
sf 12  cr 1 pl 1000 bw 125
rectime node  1    33431.552
CHECK node 1 (sf:12 bw:125 freq:8.600000e+08) others: 1
>> node 0 (sf:12 bw:125 freq:8.600000e+08)
frequency coll 125
collision sf node 1 and node 0
collision timing node 1 (0.0,98.304,33431.552) node 0 (-6366.81768946,27064.7343105)
not late enough
pwr: node 1 -120.36 dBm node 0 -121.15 dBm; diff 0.79 dBm
collision pwr both node 1 and node 0
nrCollisions  0
energy (in mJ):  5215.322112
sent packets:  2
collisions:  0
received packets:  0
processed packets:  0
lost packets:  0
NoACK packets:  0
```

EXÉCUTION DU FICHIER ONEBS



EXÉCUTION DU FICHIER MULBS

- C'est un exécutable pour la simulation avec multiples stations de base.
- **Usage :**

./MulBS.py <nodes> <avgsend> <experiment> <simtime> <datasize> <SF> <BW> [collision]

D'où :

Nodes : le nombre de nœuds.

Avgsend : le temps moyen de transsmision.

Experiment : pareil que dans OneBS

EXÉCUTION DU FICHIER MULBS

- Simtime : durée totale de la simulation en millisecondes.
- Datasize : taille du paquet à transmettre.
- SF : facteur d'étalement.
- BW : bande passante.
- Collision : 0 pour simple collision

  1 pour full collision.

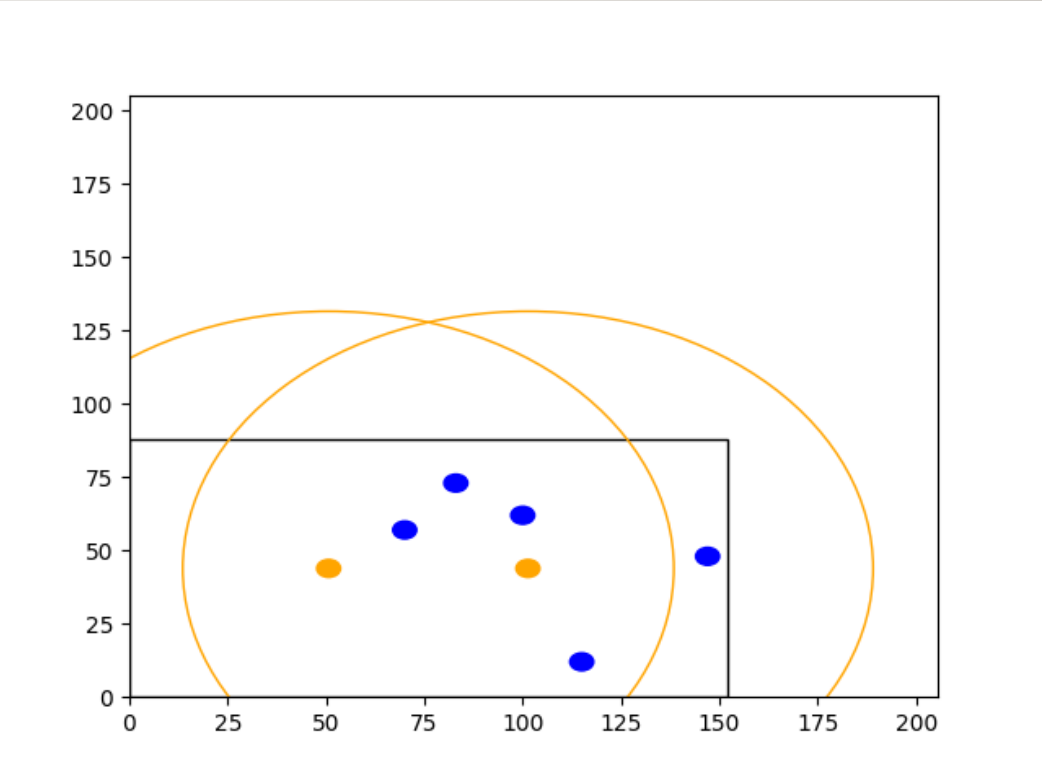
CLASS MYBS

```
class myBS():
    def __init__(self, id):
        self.id = id
        self.x = 0
        self.y = 0

        # This is a hack for now
        global nrBS
        global maxDist
        global maxX
        global maxY

        if (nrBS == 1 and self.id == 0):
            self.x = maxX/2.0
            self.y = maxY/2.0


        if (nrBS == 3 or nrBS == 2):
            self.x = (self.id+1)*maxX/float(nrBS+1)
            self.y = maxY/2.0

        if (nrBS == 4):
            if (self.id < 2):
                self.x = (self.id+1)*maxX/3.0
                self.y = maxY/3.0
            else:
                self.x = (self.id+1-2)*maxX/3.0
                self.y = 2*maxY/3.0

        if (nrBS == 6):
            if (self.id < 3):
                self.x = (self.id+1)*maxX/4.0
                self.y = maxY/3.0
            else:
                self.x = (self.id+1-3)*maxX/4.0
                self.y = 2*maxY/3.0

        if (nrBS == 8):
            if (self.id < 4):
                self.x = (self.id+1)*maxX/5.0
                self.y = maxY/3.0
            else:
                self.x = (self.id+1-4)*maxX/5.0
                self.y = 2*maxY/3.0

        if (nrBS == 24):
            if (self.id < 8):
                self.x = (self.id+1)*maxX/9.0
                self.y = maxY/4.0
            elif (self.id < 16):
                self.x = (self.id+1-8)*maxX/9.0
                self.y = 2*maxY/4.0
            else:
```

EXÉCUTION DU FICHIER MULBS

```
BSx: 50.6606883871 BSy: 43.8734431164
BSx: 101.321376774 BSy: 43.8734431164
first node
128.172350771
node 0 bs 0 lost False
126.050121133
node 0 bs 1 lost False
('node 0', 'x', 83, 'y', 73, 'dist: ', [43.52226317055279, 34.409724828801913])
135.358544982
node 1 bs 0 lost False
128.645894114
node 1 bs 1 lost False
('node 1', 'x', 147, 'y', 48, 'dist: ', [96.42764869973341, 45.864638791954086])
132.694736339
node 2 bs 0 lost False
126.121993959
node 2 bs 1 lost False
('node 2', 'x', 115, 'y', 12, 'dist: ', [71.801555658063265, 34.684594699233173])
122.55656456
node 3 bs 0 lost False
125.931489629
node 3 bs 1 lost False
('node 3', 'x', 70, 'y', 57, 'dist: ', [23.373392335684159, 33.960788251299924])
129.877414867
node 4 bs 0 lost False
120.284048861
node 4 bs 1 lost False
('node 4', 'x', 100, 'y', 62, 'dist: ', [52.563673148733905, 18.174655458420084])
ERROR: packet already in
node 2: packet will be lost
ERROR: packet already in
node 2 buffer 20 bytes
ERROR: packet already in
ERROR: packet already in
ERROR: packet already in
node 3 buffer 20 bytes
ERROR: packet already in
node 2: packet will be lost
ERROR: packet already in
node 2 buffer 20 bytes
ERROR: packet already in
energy (in mJ):  566.653157376
sent packets:  6
collisions:  0
received packets:  2
processed packets:  0
lost packets:  0
Bad CRC:  0
NoACK packets:  3
DER: 1.0
DER method 2: 0.333333333333
packets at BS 0 : 0
packets at BS 1 : 2
sent packets:  6
Press Enter to continue ...
```

EXÉCUTION DU FICHIER ONEBS



CALCUL DE L'ENERGIE

```
# compute energy
# Transmit consumption in mA from -2 to +17 dBm
TX = [22, 22, 22, 23,                                   # RFO/PA0: -2..1
      24, 24, 24, 25, 25, 25, 25, 26, 31, 32, 34, 35, 44,  # PA_BOOST/PA1: 2..14
      82, 85, 90,                                       # PA_BOOST/PA1: 15..17
      105, 115, 125]                                    # PA_BOOST/PA1+PA2: 18..20
RX = 16
V = 3.0     # voltage XXX
txpow = 14
sent = sum(n.sent for n in nodes)

energy = sum((( mynRecTime(node) * node.sent * TX[int(txpow)+2])+(node.rxtime * RX)) * V  for node in nodes)  / 1e3
```