```
#Vy Tran
#ECS 152A
#Project Part 2

import random
import matplotlib.pyplot as py_plot
from SimPy.Simulation import *

lambda_value = [0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09]
RANGE = 1000000

#the number of slots to delay of exponential backoff calculated in range 0 <= r
    <= 2**k
def exponential_backoff(retransmissions):
    return random.randint(1,2**(min(retransmissions, 10)))

#the number of slots to delay of exponential backoff calculated in range 0 <= r
    <= k
def linear_backoff(retransmissions):
    if(retransmissions == 0):
        return 1
    return random.randint(1,(min(retransmissions, 1024)))


class Packet(Process):
    #Packet arrives at host, is served (transmitted) ONLY at next slot boundary,
        leaves
    def arrive(self, res):
        yield request, self, res
        yield passivate, self
        yield release, self, res


class Source(Process):
    def generate(self, arrival_rate, Resource_process):
        i = 0

        while True:
            interarrival = random.expovariate(arrival_rate)
            yield hold, self, interarrival
            packet = Packet(name = "Packet%05d"%(i))
            activate(packet, packet.arrive(res = Resource_process))
            i += 1


class Define:
    success = 0
    collisions = 0

#define nodes
class Node(object):
    def __init__(self, lambd, backoff):
        self.source = Source()
        self.resource = Resource(capacity = 1)
        activate(self.source, self.source.generate(arrival_rate = lambd,
            Resource_process = self.resource),at=0.0)
```

```
        self.backoff = backoff
        self.retransmissions = 0
        self.slot = 0.0
        self.Choose_slot()

    def Choose_slot(self):
        if(self.backoff == "linear"):
            between = linear_backoff(self.retransmissions)
        elif(self.backoff == "exponential"):
            between = exponential_backoff(self.retransmissions)

        self.slot += between


#deal with collission state after a collision is detected
class NodeCollection(object):
    def __init__(self, lambd, backoff):
        self.nodes = [0] * 10
        for x in range(0,10):
            self.nodes[x] = Node(lambd, backoff)

    def collisions(self, RANGE):
        collideNodes = []
        for x in range(0,10):
            if(self.nodes[x].slot == RANGE):
                if(len(self.nodes[x].resource.activeQ) == 1):
                    collideNodes.append(x)

        return collideNodes


class Ethernet(Process):
    def transmit(self, nodes):
        while True:
            yield hold, self, 1
            collisions = nodes.collisions(now())

        #check for collisions
            if(len(collisions) == 1):
                reactivate(nodes.nodes[collisions[0]].resource.activeQ[0])
                Define.success += 1
                nodes.nodes[collisions[0]].retransmissions = 0

            elif(len(collisions) > 1):
                Define.collisions += 1
                for node in collisions:
                    nodes.nodes[node].retransmissions += 1
            else:
                Define.collisions += 1

            for x in range(0,10):
                if(nodes.nodes[x].slot == now()):
                    nodes.nodes[x].Choose_slot()


functions = [exponential_backoff,  linear_backoff]
```

```
exponential_array = []
linear_array = []

print("Exponential Backoff:")

#deal with collisions and successes of exponential backoff
for LAMBDA in lambda_value:
    collisions = 0
    successes = 0

    for x in range(0,10):
        Define.success = 0
        Define.collisions = 0

        initialize()
        nodeCollection = NodeCollection(LAMBDA, "exponential")
        transmission_Process = Ethernet()
        activate(transmission_Process, transmission_Process.transmit(nodes =
            nodeCollection), at = 0)
        simulate(until=RANGE)
        successes += float(Define.success)
        collisions += float(Define.collisions)

    #formula to calculate throughput
    exponential_array.append(successes/(successes + collisions))
    print("Lambda: {}".format(LAMBDA),"Successes:
        {}".format(successes),"Collisions: {}".format(collisions), "Throughput:
        {}".format(successes/(successes + collisions)))

print(" ")
print("Linear Backoff:")

#deal with collisions and successes of linear backoff
for LAMBDA in lambda_value:
    collisions = 0
    successes = 0

    for x in range(0,10):
        Define.success = 0
        Define.collisions = 0

        initialize()
        nodeCollection = NodeCollection(LAMBDA, "linear")
        transmission_Process = Ethernet()
        activate(transmission_Process, transmission_Process.transmit(nodes =
            nodeCollection), at = 0)
        simulate(until=RANGE)
        successes += float(Define.success)
        collisions += float(Define.collisions)

    #formula to calculate throughput
    linear_array.append(successes/(successes + collisions))
    print("Lambda: {}".format(LAMBDA),"Successes:
        {}".format(successes),"Collisions: {}".format(collisions), "Throughput:
        {}".format(successes/(successes + collisions)))
```

```
#plot for exponential back off
p1, = py_plot.plot(lambda_value, exponential_array)

py_plot.xlabel("Arival Rate " r'$\lambda$' "(pkts/sec)")
py_plot.ylabel("Throughput (pkts/sec)")
py_plot.title('Project 2: Exponential backoff')
py_plot.grid(True)
py_plot.savefig("exponential.jpeg")

py_plot.show()

#plot for linear back off
p2, = py_plot.plot(lambda_value, linear_array)

py_plot.xlabel("Arival Rate " r'$\lambda$' "(pkts/sec)")
py_plot.ylabel("Throughput (pkts/sec)")
py_plot.title('Project 2: Linear backoff')
py_plot.grid(True)
py_plot.savefig("linear.jpeg")

py_plot.show()

#plot for both to compare results
p1, = py_plot.plot(lambda_value, exponential_array)
p2, = py_plot.plot(lambda_value, linear_array)

py_plot.legend([p1, p2], ["Exponential Backoff", "Linear Backoff"], loc = 2)
py_plot.xlabel("Arival Rate " r'$\lambda$' "(pkts/sec)")
py_plot.title('Exponential Backoff and Linear Backoff')
py_plot.ylabel("Throughput (pkts/sec)")
py_plot.grid(True)
py_plot.savefig("compare.jpeg")

py_plot.show()

#end
```