# Haozhe Gu 999200555 & (Your Name)

# ECS 152A Project Phase II

import simpy

import matplotlib.pyplot as plt

import math

import random

import numpy as np

### Helper Function ######

def Time\_Transmission():

return random.randint(1,1544)\*Trans\_Rate

def Time\_Backoff(n):

return random.randint(1,n\*T)

class Packet(object):

def \_\_init\_\_(self,env,From,To,Collision,ACK):

self.env = env

self.Start\_Time = self.env.now

self.Collision = 0

self.From = From #Which Host create this Packet

self.To = random.randint(1,N-1) #Randomly send

if(ACK==1):#Specify ACK Receiver

self.To =To

self.Type = ACK # 1 for ACK Packet

if(self.Type==0): # Normal Packet

self.Size = random.randint(1,1544)

else: # ACK

self.Size = 64

if (self.From ==self.To):

self.To = random.randint(1,N-1)

class Host(object):

def \_\_init\_\_(self,env,ID,Lambda):

self.env = env

self.queue = list()

self.Ready = 0

self.ID = ID

self.ACKed = 0

self.Waiting\_ACK = 0

self.Failed\_ACK = 0

self.Host\_ACK\_Timeout = 0

self.Backoff = 0

self.Lambda = Lambda

self.Time\_Arrival = round(random.expovariate(self.Lambda),3)

def Check\_Ready(self):

#Check Arrival

global Total\_Packet

global Total\_Byte

global Total\_Delay

global Num\_Timeout

self.Ready = 0

if(self.Time\_Arrival<=self.env.now):

self.queue.insert(0,Packet(self.env,self.ID,-1,0,0))#-1, Normal PKT will Overwrite

self.Time\_Arrival = self.env.now+round(random.expovariate(self.Lambda),3)

#Waiting for ACK, Stop Transmitting

if(self.Waiting\_ACK==1 and len(self.queue)>=1):

if(self.Host\_ACK\_Timeout<self.env.now):

#Time\_Out for ACK

self.Failed\_ACK=self.Failed\_ACK+1

self.Backoff =Time\_Backoff(self.Failed\_ACK)

self.Waiting\_ACK =0

self.queue[len(self.queue)-1].Collision = 0 #Reset collision

Num\_Timeout = Num\_Timeout+1

elif(self.Failed\_ACK>=3):

#Abort Packet

self.Waiting\_ACK =0

self.ACKed=0

self.queue.pop()

self.Backoff = Time\_Backoff(1)

elif(self.ACKed==1):

#Got ACK in Time

self.ACKed=0

self.Waiting\_ACK =0

tmp = self.queue.pop()

self.Backoff = Time\_Backoff(1)

Total\_Packet = Total\_Packet+1

Total\_Byte = Total\_Byte+tmp.Size+64

Total\_Delay = Total\_Delay+(self.env.now - tmp.Start\_Time)

else: #Not Waiting for ACK

#Count\_Down

if(self.Backoff==0 and len(self.queue)>=1):

self.Ready = 1;

elif(self.Backoff>=1 and len(self.queue)>=1):

self.Backoff = self.Backoff-1

#else:

#print("Should not happen")

def Collision(self):

global Num\_Collision

self.queue[len(self.queue)-1].Collision = 1

Num\_Collision = Num\_Collision+1

def Transfer(self,Hosts):

Time = 0

PKT = self.queue[len(self.queue)-1]

Receiver = Hosts[PKT.To]

if(PKT.Type==0):# Normal PKT

self.Waiting\_ACK=1

self.Host\_ACK\_Timeout = self.env.now+ACK\_Timeout

Time=(DIFS+Trans\_Rate\*PKT.Size)

if(PKT.Collision==0):#No Collision

Receiver.queue.insert(0,Packet(self.env,Receiver.ID,self.ID,0,1))

else:#Collision

Receiver.queue.insert(0,Packet(self.env,Receiver.ID,self.ID,1,1))

else: # ACK PKT

if(PKT.Collision==0):

#ONLY Send ACK if no Collision

# Collision at sender make Data.Collision==1 and ACK.Collision==1

# Collision at Receiver(ACK) also make ACK.Collision ==1

Receiver.ACKed=1

Time = (SIFS+Trans\_Rate\*64)

# else:

# print("Collision!")

# Need to pop the ACK PKT

self.queue.pop()

self.Backoff = Time\_Backoff(1)

return Time

class WIFI(object):

def \_\_init\_\_(self,env,N,Lambda):

self.env = env

self.N = N

self.Hosts = [Host(env,i,Lambda) for i in range(N)]

self.Busy = 0

#self.Collision\_Count = [0 for i in range(self.N)]

self.action = env.process(self.run())

def run(self):

while True:

Ready\_Index = -1

Ready\_Num = 0

if(self.Busy==0):#IDLE

for i in range(self.N):

self.Hosts[i].Check\_Ready()

if(self.Hosts[i].Ready == 1):

Ready\_Num = Ready\_Num+1

Ready\_Index = i

if(Ready\_Num>1):#CHECK COLLISION

for i in range(self.N):

if(self.Hosts[i].Ready==1):

self.Hosts[i].Collision()

tmp = self.Hosts[i].Transfer(self.Hosts)

yield self.env.timeout(tmp)

# Collision solved via ACK

if(Ready\_Num ==1 and Ready\_Index!=-1):#Do Transfer

tmp = self.Hosts[Ready\_Index].Transfer(self.Hosts)

yield self.env.timeout(tmp)

#yield self.env.timeout(Tmp+DIFS+SIFS+64\*Trans\_Rate)

#self.Ack\_Num=Ready\_Index

#self.Waiting=SIFS+self.env.now

#else: #BUSY

yield self.env.timeout(0.001\*0.01)

#MAIN

#Initialization

Max\_Time = 20

##Sense the channel every 0.01 msec

Nlist= [10,20] #Number of Hosts

Lambdas = [0.01,0.05,0.1,0.3,0.6,0.8,0.9]

SIFS = 0.05\*0.001

DIFS = 0.1\*0.001

T = 1000

Time\_Out = [5\*0.001,10\*0.001,15\*0.001]

Trans\_Rate = 8/(11\*10\*\*6) #second per byte

ACK\_Timeout = Time\_Out[1]

Total\_Packet = 0

Total\_Byte = 0

Num\_Collision= 0

Num\_Timeout = 0

Total\_Delay = 0

#Run

print("---------------- Starting Phase II ----------------")

for i in Lambdas:

N = Nlist[1]

env = simpy.Environment()

wifi = WIFI(env,N,i)

env.run(until=Max\_Time)

print("Lambda is： "+str(i)+"N is: "+str(N)+"ACK\_Timeout: "+str(ACK\_Timeout)+"Backoff Coefficient: "+str(T))

print("Total Packet is: "+str(Total\_Packet))

print("Throughput is: "+str((8\*Total\_Byte)/Max\_Time))

print("Total Number of Collision is: "+str(Num\_Collision))

print("Total Number of Time\_Out is: "+str(Num\_Timeout))

print("Total Delay is: "+str(Total\_Delay))

print("Average Network Delay is: "+str(Total\_Delay/((8\*Total\_Byte)/Max\_Time)))