Network Analysis and Simulation - AY 2020/2021
Homework 3

Amedeo Giuliani - 2005797

May 14, 2021

Exercise 1

Our goal is to analyze the behavior of a single server queue in two cases: with service time one slot and geometric service time.
For the first case, we have that $\mathbb{P}\{1\ arrival\} = \mathbb{P}\{2\ arrivals\} = a$ while $\mathbb{P}\{no\ arrivals\}$ = 1-2a and we let vary in the interval $[0, 0.5]$. First of all, we plot the delay in function of the utilization factor ρ, which is $3a$, by varying a in the interval $[0, \frac{1}{3}]$. From what we can see in Fig. 1, the delay starts to diverge when we get close to $\rho = 1$, which is the boundary of the stable region.

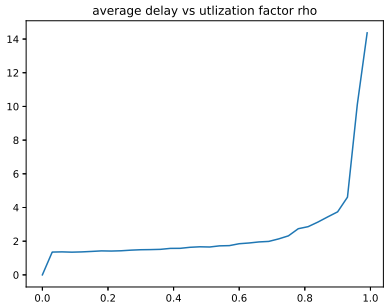


Figure 1: Delay vs utilization factor ρ

Then, if we analyze the queue size over time 2, with $a \in \{\frac{1}{4}, \frac{1}{3}, \frac{1}{2}\}$, we can tell again that: $a = \frac{1}{4}$ is inside the stable region because the queue size is on average constant; $a = \frac{1}{3}$ is the boundary of the stable region since the queue size starts to get larger but it still remains bounded; $a = \frac{1}{2}$ is outside the stable region as the queue size is unbounded and increases linearly with time.

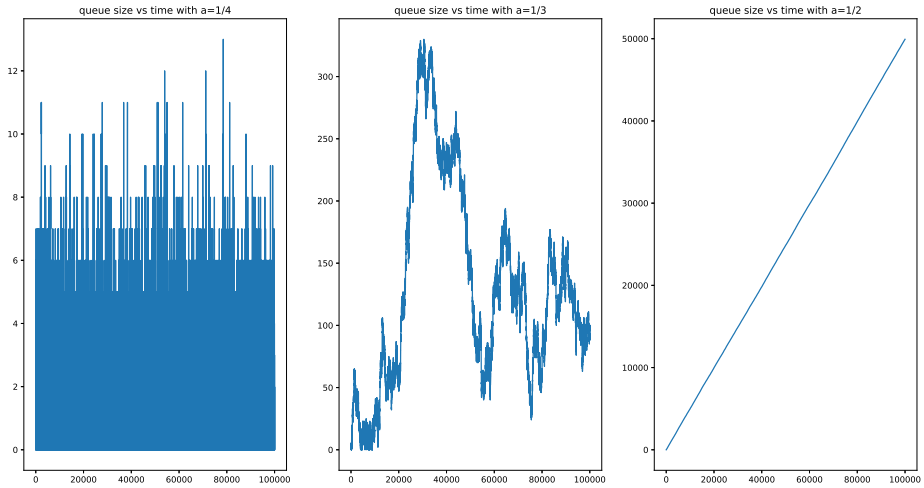


Figure 2: Queue size vs time

We now consider a queue with geometric service time with mean $\frac{1}{b}$ and $\mathbb{P}\{1\ arrival\} = 1 - \mathbb{P}\{no\ arrivals\} = a = 0.5$. This time, the utilization factor is $\rho = \frac{a}{b}$ and in Fig. 3 is shown the evolution of the delay in function of the first, with $b \in [0.5, 1]$. Of course, the delay starts to diverge in proximity of the boundary of the stable region, i.e. for values of b close to 0.5.

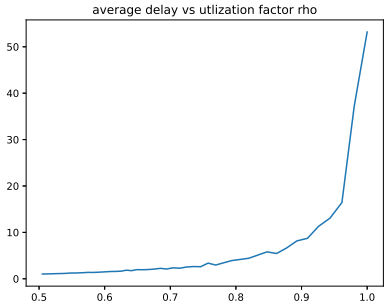


Figure 3: Delay vs utilization factor ρ

We also want to characterize the queue size over time with $b \in \{\frac{1}{3}, \frac{1}{2}, \frac{2}{3}\}$. From Fig. 4 we can clearly see that: $b = \frac{2}{3}$ is in the stable region as the queue size is on average constant; $b = \frac{1}{2}$ is the boundary of the stable region as the queue size gets larger but it still remains bounded; $b = \frac{1}{3}$ is in the unstable region and in fact the queue size is unbounded and increases linearly with time.

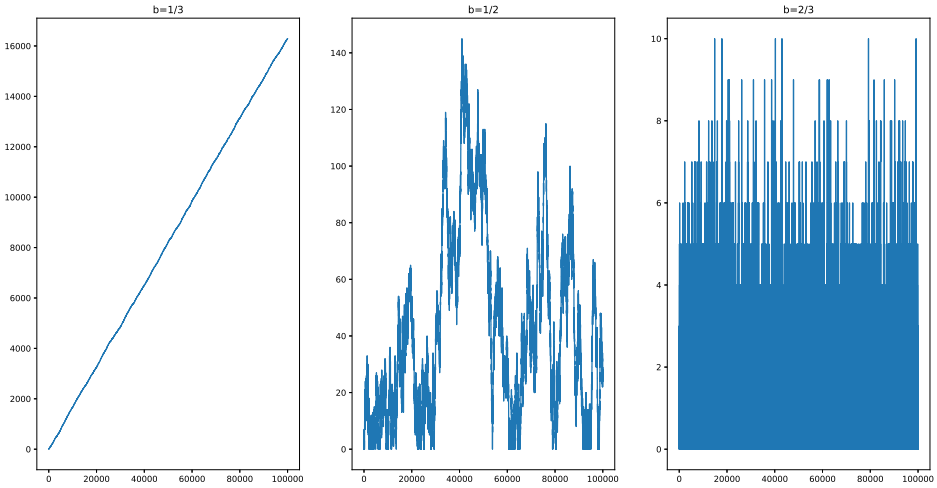
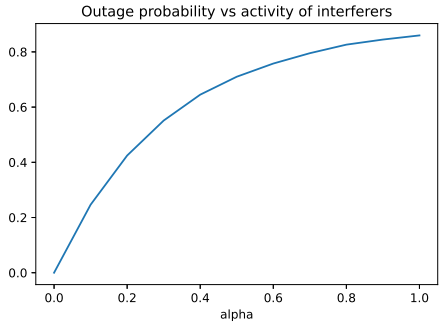


Figure 4: Queue size vs time

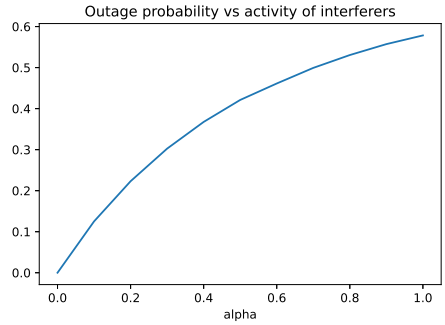
Ultimately, we want to find the queue size for which $\mathbb{P}\{overflow\} = 0.001$. In the first case, for $a = 0.25$ the requirement is met with a queue size of 10 slots, while for $a = \frac{1}{3}$ we have a queue size of 153 slots. In the second case, for $b = \frac{2}{3}$ the requirement is met with a queue size of 10 slots, while for $b = \frac{1}{2}$ we have a queue size of 88 slots. Of course, this last result is less than what we obtained in the first case because here the service time is geometric and therefore one user spend on average more time in the system with respect to the case with service time 1 slot.

Exercise 2

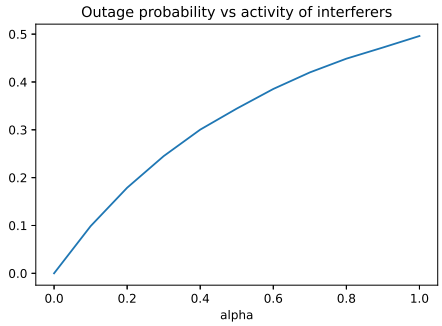
We want to analyze the outage probability in a cellular scenario with six interfering cells and the performances of the slotted ALOHA through Monte-Carlo simulations. Let us approximate the user cell with the circumference inscribed in the hexagon, that is, a circle with radius $r = 0.91$ and the interfering cells with a segment of circular ring. Moreover, let $N = \{1, 3, 4, 7\}$, $b = \{6, 10\}$ dB, $\eta = 4$ and $\sigma_\psi = 8$ dB be: the reuse factor, the *Signal to Interference plus Noise Ratio* (SINR) threshold, the path loss factor and the standard variation of the shadowing, respectively. We ran our simulations for every combination of the previous parameters.



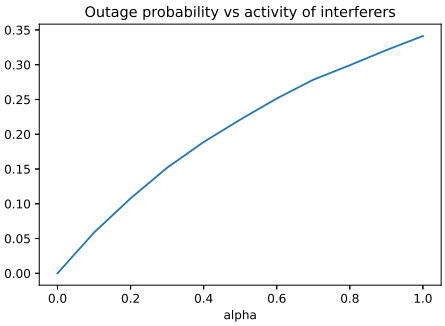
(a) $N = 1$



(b) $N = 3$



(c) $N = 4$



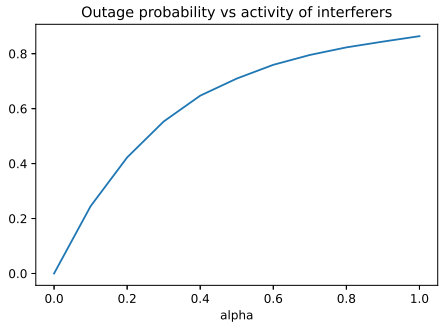
(d) $N = 7$

Figure 5: $b = 6$ dB

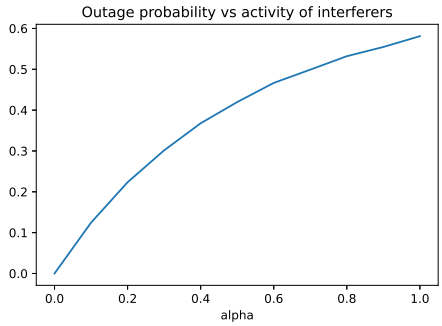
The outage probability Pout in function of $\alpha \in [0, 1]$, the probability to use an interfering channel, is shown in Fig. 5 for $b = 6$ dB and in Fig. 6 for $b = 10$ dB. In both cases the outage probability will be lower with increasing N: this is of course expected, as the interfering cells, which reuse the same frequency used in the user cell, will be further from the latter as the reuse distance $D = \sqrt{3Nr}$ will increase. We also notice that with the second threshold the outage probabilities are higher than with the first threshold. This is again expected as with increasing b the condition to determine an outage event becomes looser.
Now, we proceed with the simulation of the capture probability, for which we fixed $N = 7$. From Fig. 7 we can see that is a decreasing function of the number of collisions $n \in [2, 30]$, as it should be, since the ALOHA protocol does no perform well when we have a high number of users of a high traffic in the network. Ultimately, to study the throughput, once we obtained the vector of capture probabilities for each value of n, we used the following formula:

$$S(G) = \sum_{n=2}^{+\infty} \frac{G^n e^{-G}}{n!} C_n \tag{1}$$

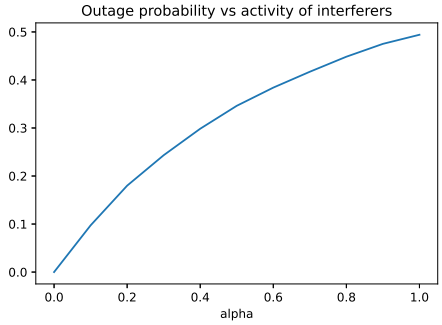
and the result is shown in Fig. 8. However, for some reasons we were unable to detect, the curve of the throughput does not go down as G increases thus forming the classical bell shape, but the tail remains "opened".



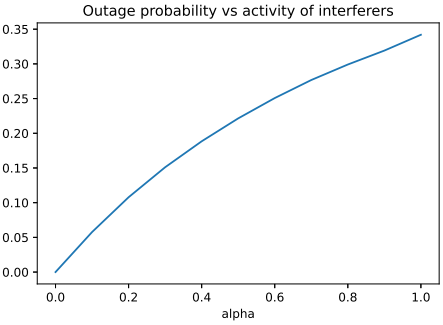
(a) $N = 1$



(b) $N = 3$

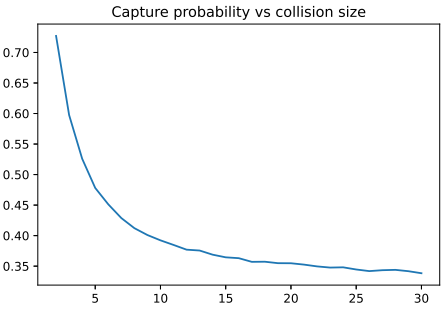


(c) $N = 4$

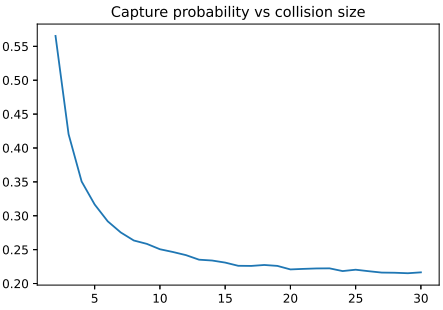


(d) $N = 7$

Figure 6: $b = 10$ dB

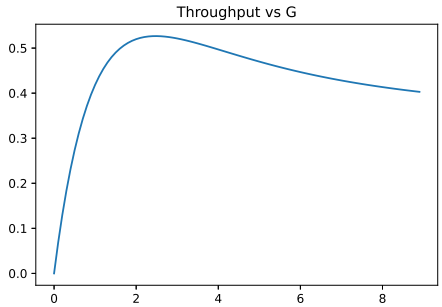


(a) $b = 6$ dB

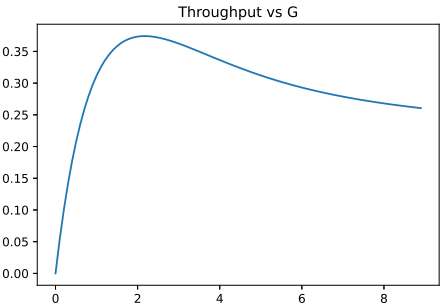


(b) $b = 10$ dB

Figure 7



(a) $b = 6$ dB



(b) $b = 10$ dB

Figure 8

Exercise 3

To study the multi-hop performance of *Geographic Random Forwarding* (GeRaF) we developed the following code in Python:

```
def makeHop(D, M):
global xmin
global ymin
x = []
y = []
n = np.random.poisson(M)
if n == 0:
return D
# generate n uniform samples within a circle centered in (xmin,ymin)
for i in range(n):
r = np.sqrt(np.random.uniform(0, 1))
theta = np.random.uniform(0, 1, 1)*2*np.pi
x = np.append(x, xmin+r*np.cos(theta))
y = np.append(y, ymin+r*np.sin(theta))

dist = np.sqrt(np.power(x, 2)+np.power(y, 2))
mindist = np.min(dist)
idx = np.argmin(dist)
xmin = x[idx]
ymin = y[idx]
D = mindist

return D

def nHops(D, M):
global xmin
global ymin

xmin = D
ymin = 0
hops = 1
while(D > 1):
d = makeHop(D, M)
if d != D:
hops += 1
D = d
return hops

xmin = D
ymin = 0
M = 30
K = 100
hops = np.empty((0,M),int)

for j in range(K):
        h = []
        for m in range(M):
                c = nHops(D,m+1)
                h = np.append(h,c)

        #stack up all the K repetitions of the experiment
        h = h.reshape(-1,M)
        hops = np.append(hops,h,axis=0)

#perform mean and std among the K repetitions of the experiment
avg_hops = np.mean(hops,axis=0)
```

```
std_hops = np.std(hops,axis=0)
```

which gives as results Fig. 9, Fig. 10 and Fig. 11 with D equal to 5, 10 and 20, respectively.

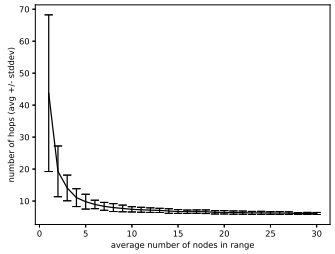


Figure 9: D = 5

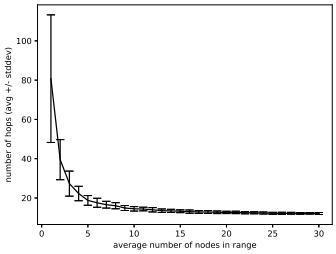


Figure 10: D = 10

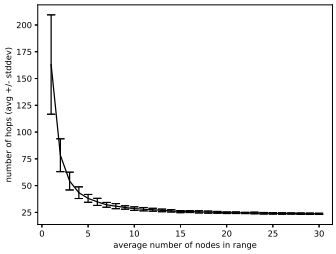


Figure 11: D = 20