# CS 350 – Fall 2020, Assignment 1

**Answer 1**

a) Considering a MTBF of 5 minutes, and a reboot time of 1 minute, then we know that the server will be offline (or rebooting) 1/6th of the time, and since a request is only lost if at receiving time the reboot is happening, then we know that **the probability of a request being lost is 0.17.**

b) Since we know that the probability of any given request being lost is 1/6, then **every 6th request will, on average, be lost.**

c) Since we know a new request is interrupted if at any point during its two minute complete time, then the probability that a new request will be complete is $R(2) = e^{-2/5} = 0.67$ **or 67%**.

d) If we work back from the probability of the server being up for x amount of time, we can quickly figure out that: $0.95 = e^{-x/5} \equiv x = 0.25$ or x = 15 seconds. With that in mind, if we divide 2 minutes (the service time) by 15 seconds, we get that: $120/15 = 8$, as such, we can state that **with N=8, 95% of all request would be successful processed.**

e) Given $N = \infty$ (ie: all request will eventually be processed), and since we know that the MTBF is 5 minutes, and that the probability of a single request being completed first time around is 67%, we know that more two thirds of the requests will be completed the first time around. However, we can calculate based on the MTBF that 1 in every 3 requests will require a second attempt, on average of course, **so we have that the average N is** $0.67 + 0.33 \times 2 = 1.33$.

f) **Yes, because although it is true that the server is more likely to stay up for the time needed to process half of the request, since the other half is processed right after, and if it fails the entire request fails, then independently of how likely the server is to stay up for half, it still has the same probability to stay up for the entire required time to complete the request.**

g) If we work backwards in the math we did for d, we get that for $120/x = 3 \equiv x = 40$, so now we need to know the probability of the server staying up for 40 seconds so $p = e^{-40/300} = 0.875$. Now if we apply this math to the new style, we need to calculate the probability of the system staying up 1 minute with 2 attempts so $60/x = 2 \equiv x = 30$, so $p = e^{-30/300} = 0.904$, however, we need to know the probability of this happening TWICE so, $P(A\ \&\ B) = 0.904^2 = 0.817$, so no, **N=2 in the new formula is not enough to match N=3 in the old one**, to match the old one you'd still need N=3.

**Answer 2**

a) With an average inter-arrival time of 45 seconds, we have that the throughput of the system is 1.3 events per minute.

b) The function for the probability distribution of the inter-arrival times for CodeBuddy is a Poisson Distribution, and, as such, as a formula of $f(x) = \frac{0.75^x}{x!} \times e^{-0.75}$ where x is the minute in question.

c) Due to the memoryless property of the Poisson Distribution, the probability is the same as the probability of the first even occurring 1 minute after the start time so $f(1) = \frac{0.75^1}{1!} \times e^{-0.75} = 0.354$.

d) Since the standard deviation of a poisson property is the square root of the mean we have that the standard deviation of this distribution: $\sqrt{0.75} = 0.886$.

e) First we need to know, on average, how many request arrive in 120 seconds $120/45 = 2.6$, now we can adapt the Poisson distribution $g(x) = \frac{2.6^x}{x!} \times e^{-2.6}$

f) If we apply the CDF to the formula above to check the probability of less than 3 events reaching the system in the 120 seconds and then subtract it from 1, we get the probability of more than 3 events reaching the system, so CDF $P(x \leq 3) = F(3) = \sum_{i=0}^{3} \frac{2.6^i}{i!} \times e^{-2.6} = 0.736$, so to get $P(x > 3) = 1 - F(3) = 0.264$.

g) The utilization of CodeBuddy is $20/30 = 2/3 = 0.666$ or $66.6\%$.

h) On average, since CodeBuddy has a response time of 30 seconds, and an inter-arrival time of 45 seconds, then this number would, on average, be 0.

**Answer 3**

a) Considering that in steady-state the system is handling 14000 active requests, than any new request must inherently wait for the other 14000 to complete, since it's the steady state we can admit that at this point whenever a request enters, another one leaves, so since requests are entering at a rate of 4.2 per millisecond ($4200/1000 = 4.2$), then that has to be the rate at which they leave, hence it will take approximately 3.3 seconds ($\frac{14000}{4.2}/1000 = 3.3$) for a single request to be processed, as such the average latency for a single request is 3.3 seconds.

b) Theoretically speaking, a very light load for this server could have a latency of just 3ms, if the requests arrive a a slower rate than they are resolved. If this was the case, when compared to the latency of 3.3 seconds, we have a total slowdown of 110000% under the heavy load (Rule of 3: if 3ms = 100% then 3300 = 110100%).

c) As we've seen before, in Steady-State the System is finishing 4.2 requests per ms, since each request takes 3ms to finish, then a single CPU finishes 0.(3) request per ms, so, to finish 4.2 we would need, at least, 13 CPUs (the math ends up in 12.6 but since we can't have a fraction of a CPU, we need to round up).

d) As mentioned above, we only, theoretically, need 12.6 CPUs, but since we can't have a fraction of a CPU, we'd have 13. As such, and if the system needs to handle 4.2 requests per ms, then each CPU would handle 12.6 requests per 3ms, which when divided by 13 is 0.969 per CPU, which means that the CPU would have an utilization of 96.9%.

e) Since the system is handling 4.2 requests per ms, it means it's handling 4200 requests per second, which if we divide by 13, means that each CPU is handling 323.077 requests per second.

f) The main assumption is that the rate of new requests stays stable throughout the utilization, in addition I assumed no system-outages would occur or any other events that might affect the system performance. I also assumed that the system already started on steady-state, and would never leave it. Finally I assumed that "very light load" meant a load lower than the time it'd take for the system to process a request.