Technical University of Munich, Chair of Communication Networks
Prof. Dr.-Ing. W. Kellerer, Alberto Martínez Alba, Mu He

Analysis, Modeling and Simulation of Communication Networks
SS 2018

Programming Assignment - Part 3

Submission Deadline: 22. May 2018

3 Generating Random Numbers (40 Points)

Until now, you have been using a uniformly distributed serving time and a constant inter-arrival rate. Now, you add the possibility to draw numbers from specified distributions.

3.1 Implementation

You will implement a random number generator (RNG) for exponential and uniform distributions. A series of random numbers is called random number stream (RNS). The RNG class contains two RNS, one for the inter-arrival time and one for the service time.

Task 3.1.1: Random Number Generation

5 Points

Implement all methods in the subclasses of the abstract class *RNS*. The subclasses *ExponentialRNS* and *UniformRNS* should be initialized with a seed and all necessary parameters. The method *next()* should return a new sample of a given distribution. You can use the inverse transform method to generate new samples of a distribution. The seed input parameter should be optional. Note: You cannot use the built-in functions of the Random class to directly generate the two streams.

Task 3.1.2: Integration of the RNG

5 Points

Now, integrate your RNG into the simulation. For this, you have to modify several methods.

- First, modify the method *do_simulation()* in your class *Simulation*. Uncomment the import statements and the generation of an RNG. You should modify the generation in order to receive an RNG with two exponential random number streams. The mean of the inter-arrival time distribution should be 1s, the mean of the service time distibution should be depending on the value *Simparam.RHO*. Note, that you have to reset the RNG when you change the value of ρ!

- After that, modify all lines in the *process()* functions in the classes *CustomerArrival* and *ServiceCompletion*. You should modify at least three lines and call the dedicated RNS instead of a normal random number generation or the constant inter-arrival time.

3.2 Verification

Write your code for the following tasks in file *part3_simstudy.py*. For checking the correct implementation of the system utilization and the usage of rho, you can run the test in *part3_tests.py* first.

Task 3.2.1: Verification of distributions

5 Points

Create two histograms, one for each implemented distributions. Generate a sufficient number of samples by using the classes RNG and/or RNS and choose a reasonable number of bins for your histograms.

Task 3.2.2: Verification of system utilization

10 Points

Since you are now able to generate exponential distributions, verify your implementation of the system utilization by simulating an M/M/1/S system with a limited queue capacity (S = 5) and $\rho = \lambda/\mu = 0.01, 0.5, 0.8, 0.9$. Simulate your system for 100s and 1000s. Note, that depending on your implementation you may have to reset your simulation object when changing the ρ in the parameter object.

3.3 Analysis and General Questions

Answer the following questions separately and in full sentences. Explain your answers.

Task 3.3.1: Seed for Random Number Generation

5 Points

Why does it make sense to use a seed when simulating a non-deterministic system?

Task 3.3.2: Bins for Histograms

5 Points

How does the shape of the histograms change if you reduce/increase the number of bins? What would be a reasonable number of bins and how does it depend on the number of samples?

Task 3.3.3: System Utilization

5 Points

How do you interpret the results from task 3.2.2? What is the expected system utilization, if you simulate for an infinite simulation time? What happens, if S becomes unlimited?

Total: 40 Points