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 2

Submission Deadline: 15. May 2018

2 Extending the DES (125 Points)

In this part, you will extend your simulator, so that you can do some measurements of the system in the simulation.

2.1 Packets

Until now, the queue has been a simple counter. In order to do measurements on the delay of packets, it is necessary to extend our previous implementation and store actual packets in our system. The module *Packet* in file *packet.py* represents a packet. A packet object has eight attributes. An attribute for its arrival time, an attribute for the time when the serving of the packet starts, an attribute for the time when the serving of the packet completes, a flag indicating whether the packet has to wait, a flag indicating whether the packet is currently being served, and a flag indication whether the packet processing is completed. Besides, it contains a reference to the current simulation and the inter-arrival time to the last packet.

Task 2.1.1: Variables of a Packet

10 Points

Implement the given functions. They should modify the variables of a packet, if needed. Think about which function has to modify which variable. Note, that the system time of a packet is defined as the whole time that the packet stays in our system, i.e., the summation of its waiting time and serving time.

2.2 Finite Queue

Now, modify your system, that you have a queue which actually contains packet objects. For this, you have to implement and integrate the class *FiniteQueue* in file *finitequeue.py*. The *FiniteQueue* class is part of your queuing system. An object of the *FiniteQueue* class stores all the queued packets. The finite queue class provides methods for adding packets, removing packets, getting the current queue length, indicating whether the queue is empty, and flushing the queue, i.e., deleting all packets inside. Use Python Queue data structure for the implementation of the *FiniteQueue*.

Task 2.2.1: Implementation of a finite queue

10 Points

Implement the missing methods in the *finitequeue.py* file. Access the capacity of the queue by referencing the parameter of your simulation.

Task 2.2.2: Integration of the finite queue class

10 Points

Modify your module *SystemState*, such that the queue of your system is no longer a simple counter, but a finite queue that contains packets. Adapt your module, so it now contains a variable called *buffer* of type *FiniteQueue* instead of the counter variable. Furthermore, create a variable *served_packet*, that represents the currently being served packet. Adapt all methods that interact with the queue and modify them in the correct way. The functions *add_packet_to_server()* and *add_packet_to_queue()* should create a new packet instance once they are called. Think, which other methods have to modify variables of a packet, e.g., call functions of a packet like *complete_service*. There are no indications where to implement this method, since you have to modify almost all methods in class *SystemState*.

You should also add a new variable into the class *SystemState*: the new variable, called *last_arrival*, should indicate the last packet arrival before the current packet, so that you can calculate the inter-arrival time. You can initialize it with the value 0.

Furthermore, add a function *get_queue_length()* to class *SystemState* that returns the current size of the queue, same as the equally named function in class *FiniteQueue*.

2.3 Counter

The counter module (file *counter.py*) contains an abstract class *Counter*, a class *TimeIndependentCounter* and a class *TimeDependentCounter*. *TimeIndependentCounter* and *TimeDependentCounter* inherit from *Counter*. Both inheriting classes should implement the methods as specified by *Counter*. Use methods provided by *numpy* package if possible to easily get mean, variance, and standard deviation of a set of sample data.

Task 2.3.1: Implementation of TIC

5 Points

Implement the class *TimeIndependentCounter* with all its methods, except the two methods that are related to confidence intervals. Think carefully about the variance calculation, assuming you only have a limited number of samples.

Task 2.3.2: Implementation of TDC

10 Points

Implement the class *TimeDependentCounter* with all its methods. Note, that you can't use standard *numpy* methods for statistics calculation here any more. Hint: For the variance calculation, use the formula of raw moments. You can add any new variable that you may require.

2.4 Histogram

In order to visualize data, it makes sense to plot histograms. Histograms contain a given number of bins, which depends on the input data and the number of samples. We have provided a class *Histogram* in the file *histogram.py*. The structure of the classes in this file is similar to the counter implementation, having an abstract class *Histogram* and time independent and time dependent realizations. Adding values to an instance of the class *Histogram* is done similarly as adding values to an instance of the class *Counter*. Histograms can be plotted with the given function *plot()*. Note, that the *plot()* function is part of the abstract module *Histogram* and is called at the end of the *report()* function.

Task 2.4.1: Creating a Histogram of Time Independent Values (TIH)

10 Points

For the TIH, you have to modify the methods *count()* and *report()*. The method *count()* should add values to the internal array. Some parts of the function *report()* have been implemented already. Your task is to generate the variables: *self.histogram* and *self.bins*. You can use the given function *numpy.histogram()* and modify some of the input parameters. Refer *numpy* documentation to see how to use it.

Task 2.4.2: Creating a Histogram of Time Dependent Values (TDH)

5 Points

For the TDH, you have to modify the functions *count()* and *reset()*. Note, that now each value should be weighted by its duration. For this, you have to add values to the array *weights* as well.

Task 2.4.3: Integration of Counters and Histograms in Simulator

5 Points

The class *CounterCollection* in the file *countercollection.py* contains some predefined counters and histograms. It provides the methods *count_packet()* and *count_queue()*. The first method should be called whenever the service of a packet is finished, e.g., in the function *complete_service()* of class *SystemState*. For an easy implementation, the second method *count_queue()* can be called after each time update step, i.e., in *do_simulation()* in the class *Simulation*.

Task 2.4.4: Creating a Side-by-Side Plot

10 Points

Side-by-Side plot is useful in comparing results of different input parameters. Create one figure for the bar plots side-by-side. For this, you have to modify the function *plot()* in class *Histogram* and modify the side-by-side section. For plotting the bars side by side please refer to matplotlib examples, e.g., `https://matplotlib.org/2.1.2/gallery/statistics/barchart_demo.html`. In this online example, the bars of “Men” and “Women” are plotted side-by-side for each “Group”.

2.5 System Utilization

Your simulator should be able to measure the system utilization. Add all necessary lines to the function *count_queue()* in class *CounterCollection*.

Task 2.5.1: Measuring the System Utilization

5 Points

Think about how to measure the system utilization with the given time dependent counter *cnt_sys_util* in class *CounterCollection* and implement it.

2.6 Verification

Again, we provide you a python unittest, which checks the basic functionality of your code. The unittests in file *part2_tests* check the functionality of the different extended SystemState, FiniteQueue, Packet and the Counters. Furthermore, it can run whole simulations with the seed ranging from 0 to 5, to give you the ability to check the correct implementation of your extended DES.

When you run a simulation with a maximum queue size of 4 for 100 seconds, the system should count roughly between 5 and 20 dropped packets. Try different seeds to check, if this is the case in your simulation. Run all unittests and make sure, they do not return any errors.

2.7 Simulation Study II

Now that you have implemented your extended DES, it's time to perform some analysis. Do the coding in file *part2_simstudy.py*.

Task 2.7.1: Systems with Different Queue Capacity I

15 Points

Create plots of the mean waiting times of a packet and the mean queue length (buffer fill status). Perform this simulation for queue capacity $S = 5, 6, 7$ and simulation time of 100s. Run your simulation 1000 times and add the averages to respective histograms. Plot your histograms. You should have three histograms for the waiting times and three for the queue lengths. For easier comparison, you can also display the results for the different queue capacities in one plot only (for the mean queue length, you can use the side-by-side plot for better comparison). Think of a reasonable number of bins. Hint: You can and should use additional configuration parameters defined in class *SimParam*, like *S_VALUES*. For better performance you can also temporarily comment all unused counters in class *CounterCollection*.

Task 2.7.2: Systems with Different Queue Capacity II

5 Points

Create the same plots for a simulation time of 1000s. Still, simulation has to run 1000 times.

2.8 Analysis and General Questions

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

Task 2.8.1: Systems with Different Queue Capacity

10 Points

Describe the observations, that are highlighted by the plots from tasks 2.7.1 and 2.7.2. What can you observe, when you increase the queue capacity? What are the differences between a simulation time of 100s and 1000s, especially regarding the variance? Why does this effect happen?

Task 2.8.2: Choosing the right number of bins

5 Points

Explain the numbers of bins you choose for the histograms. Write down all your considerations.

Task 2.8.3: Variance Calculation

10 Points

In task 2.3.1 you have calculated the variance. What is the difference between the variance calculated in this manner and the variance of infinite number of samples? Where does the difference come from?

Total: 125 Points