Extra Credit Assignment (35 points)

Anirudh Sivaraman

2021/11/05

Please read the notes for lecture 12, especially the portion on input queueing, before starting this assignment. In this assignment, you'll implement the FIFO and PIM matching algorithms for maximal matching in an input-queued router. You have been given some starter code and a simulator for both algorithms (fifo.py and pim.py), which you need to develop into full implementations. We'll only be dealing with uniform traffic when simulating and understanding the performance of both FIFO and PIM: i.e., at every tick, a packet independently arrives at every input port with an identical probability p, and the packet is destined to one of the output ports picked uniformly at random. For the purpose of this assignment, we'll assume that packets coming in on an input port can also go out on the output port corresponding to that same input port. Loopback in this manner is not common as we discussed in lecture 12, but it will simplify our code.

1 Overview of starter code

The starter code consists of two files: fifo.py and pim.py. Each takes three common arguments from the user:

1. The number of input/output ports in the router. The number of input and output ports are always equal in our simulations, so you should only specify one number here.
2. The independent probability p of arrivals at each router input port.
3. A random seed to make the simulations deterministic and debuggable. This can be any integer.

In addition, pim.py takes a fourth argument representing the number of iterations of the PIM algorithm.

After reading in these user-supplied arguments, the simulation script in each case either initializes a set of FIFOs for each input port (fifo.py) or a set of virtual output queues (VOQs), i.e., one FIFO for each input-output combination, for each input port (pim.py). It then runs a simulation for 20000 ticks. You can change this if you want by adding a simulation_ticks value when you create your PimSimulator. On each tick, the simulator generates a packet at each input port with a probability p and sets its output port uniformly at random to one of the N output ports.

It then runs either the FIFO or PIM algorithm. You have to implement both for this assignment. As part of running the algorithm, both fifo.py and pim.py also compute the average delay of all packets in the simulation and print this every 100 ticks. If this average delay reaches a steady value as the simulation's tick time evolves, then the system has not saturated. If on the other hand, it keeps increasing without bound, it is an indication that the system has saturated, i.e., the queues have started growing unboundedly.

2 Running the code

Here's how you run the pim.py and fifo.py files:

```
python3 pim.py -p 16 -a 0.5 -s 1 -i 1
```

This runs the PIM algorithm on a 16x16 router with an arrival probability of 0.5, a random seed of 1, and 1 iteration of the PIM algorithm.

The output of this command prints out the current average packet delay every 100 ticks, averaged over all packets since tick 0. Here is a snippet of the output from the command above.

```
Anirudhs-MacBook-Air:asg4e_code_sols anirudh$ time python3 pim.py -p 16 -a 0.5 -s 1 -i 1
Average delay after  0  ticks =  0.0  ticks
Average delay after  100  ticks =  1.4682151589242054  ticks
Average delay after  200  ticks =  1.4314320388349515  ticks
Average delay after  300  ticks =  1.5203748981255094  ticks
Average delay after  400  ticks =  1.4373284537968893  ticks
Average delay after  500  ticks =  1.3608094768015795  ticks
Average delay after  600  ticks =  1.3709345409633593  ticks
Average delay after  700  ticks =  1.3567634560906516  ticks
Average delay after  800  ticks =  1.3738187451587915  ticks
Average delay after  900  ticks =  1.4194217734126435  ticks
```

Similarly, you can run fifo.py with the same arguments, except that the FIFO algorithm does not take an argument for the number of iterations.

```
Anirudhs-MacBook-Air:asg4e_code_sols anirudh$ python3 fifo.py -p 16 -a 0.5 -s 1
Average delay after  0  ticks =  0.0  ticks
Average delay after  100  ticks =  1.0411311053984575  ticks
Average delay after  200  ticks =  1.6926020408163265  ticks
Average delay after  300  ticks =  1.9262192580241768  ticks
Average delay after  400  ticks =  1.8090202177293935  ticks
Average delay after  500  ticks =  1.8579234972677596  ticks
Average delay after  600  ticks =  1.7829084830756372  ticks
Average delay after  700  ticks =  1.7504939823962637  ticks
Average delay after  800  ticks =  1.7795522154376076  ticks
Average delay after  900  ticks =  1.8499651081646895  ticks
Average delay after  1000  ticks =  1.8769288671433948  ticks
```

3 Testing your code

Look at the instructions in the TODOs in both pim.py and fifo.py to implement both algorithms. You can check that your PIM and FIFO implementations are correct, by confirming several things below.

1. For any given arrival probability, the average delay of PIM with a single iteration should be lower than FIFO. For instance, for a 16x16 router with an arrival probability of 0.5, the output above shows that the average PIM delay (with 1 iteration) after 900 ticks is 1.42 ticks, while it is 1.85 ticks for FIFO.

2. Similarly, all else being equal, the average delay of PIM with multiple iterations should be better than the average delay of PIM with a single iteration. For instance, on a 16x16 router with a 0.5 arrival probability, after 900 ticks, the PIM delay with a single iteration is 1.42 ticks (as shown above). But for PIM with 4 iterations, this number is 0.73 ticks (not shown above).

3. Generally speaking, your FIFO implementation should be considerably faster than PIM, and the runtime of the PIM implementation should increase with the number of iterations because you are running the same PIM algorithm multiple times. On my laptop, for a 16x16 router, with a 0.5 arrival probability, and 20000 simulation ticks, it takes about 3 seconds for FIFO, about 8 seconds for PIM with one iteration, and about 18 seconds for PIM with 4 iterations.

4. FIFO should saturate, i.e., the average delay should keep increasing as simulation time increases, once the

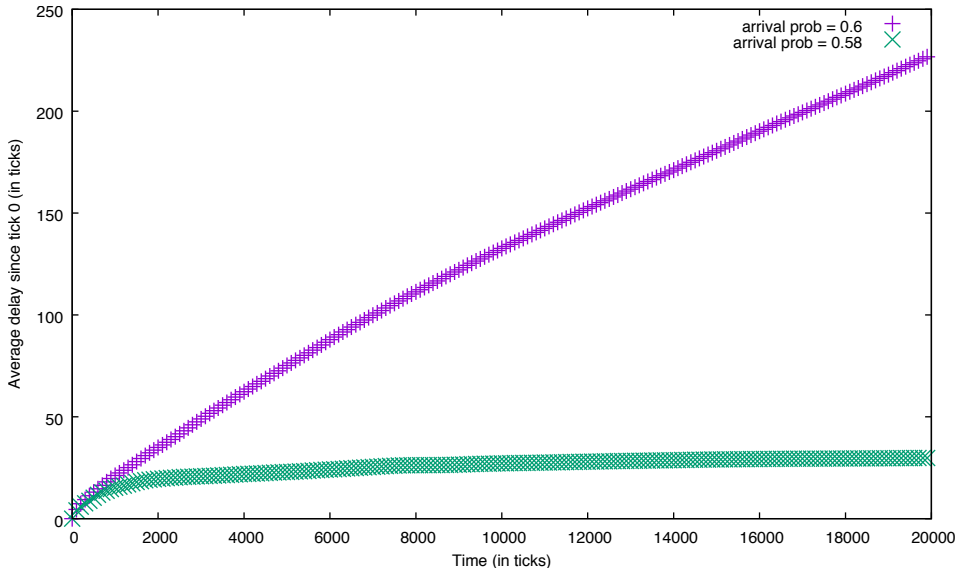

Figure 1: FIFO saturates at an arrival probability of 60%, but does not saturate at an arrival probability of 58%. This experiment had 100 input and output ports.

arrival probability crosses about 59% for a router with a large number of input and output ports.[1] You can set the number of ports to 100 for this test. You can confirm that something is saturating by plotting the average delay as a function of time and seeing that it is continuosly increasing (Figure 1). The raw data for this plot is just the output of the simulator, i.e., the average delay in ticks that is printed out every 100 ticks.

5. PIM with a single iteration should saturate once the arrival probability crosses about 63%. You can follow a similar procedure as the one above to determine if a system has saturated. The time-series plot showing that PIM saturates is shown in Figure 2.

4 Point breakdown

The point breakdown is as follows:

1. Implement the FIFO algorithm correctly, i.e., make sure that the average delay at the end of the simulation goes up with the arrival probability, and that it saturates at around 58% (12 points).

2. Implement one iteration of the PIM algorithm correctly, i.e., make sure that the average delay at the end of the simulation goes up with the arrival probability, and that it saturates at around 63% (13 points).

3. Implement a version of PIM that performs multiple iterations correctly, i.e., make sure that the average delay at the end of the simulation goes up with the arrival probability, and that it never saturates as long as the arrival probability is below 100% (10 points).

5 Miscellaneous advice (or) helpful code snippets

1. To pick an element uniformly at random from a non-empty list, e.g., to pick a random input port for an output port in the FIFO algorithm, or to pick a random input port to grant to in the PIM algorithm, you

[1] The saturation threshold increases as you decrease the number of ports. You can mathematically calculate the right threshold for an arbitrary number of ports. But, we'll keep things simple by only looking at the saturation behavior for a large number of ports.

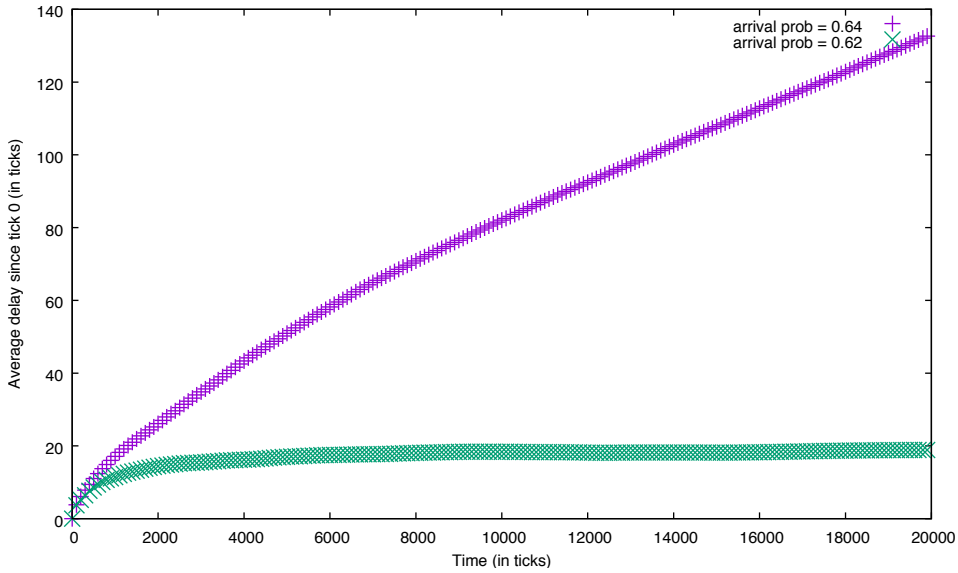

Figure 2: PIM saturates at an arrival probability of 64%, but does not saturate at an arrival probability of 62%. This experiment had 100 input and output ports.

can use the random.choice() function in Python.

2. random.choice() will throw an error when you pass it an empty list.

3. To dequeue either from an input queue (FIFO) or a virtual output queue (PIM), because that particular input queue or VOQ has been picked as part of the matching process, you can use the following code snippet.

   ```
   q = q[1:]
   ```

   Essentially, this sets q to the slice of the current q from the 2nd element onwards, which is equivalent to dequeueing.

4. To track the average delay of dequeued packets, you can use this code snippet:

   ```
   delay = tick - chosen_packet.arrival_tick
   self.delay_sum += delay
   self.delay_count += 1
   ```

   This way, the delay at any instant is `delay_sum / delay_count`.

5. If you don't update self.delay_sum and self.delay_count, you'll get the following divide-by-zero error:

   ```
   Anirudhs-MacBook-Air:asg4e_starter_code anirudh$ python3 fifo.py -p 16 -a 0.5 -s 1
   Traceback (most recent call last):
     File "fifo.py", line 57, in <module>
       print ("Average delay after ", tick, " ticks = ", delay_sum / delay_count, " ticks")
   ZeroDivisionError: float division by zero
   ```