ECE 358 Lab 1

Henry Quan and Jeffrey Seto

Table of Contents:

Question 1: Generator Results

Actual results
lamdba: 75.000000
mean: 0.012902
variance: 0.000159

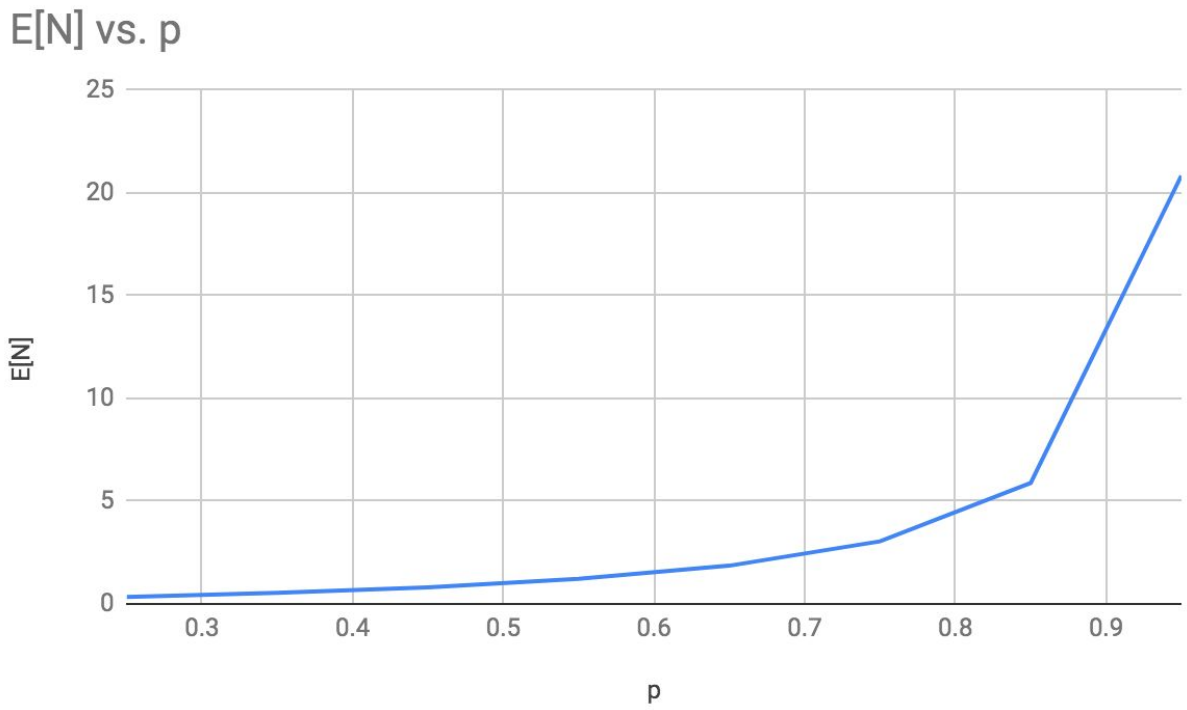
Expected results
Lambda: 75
Mean: 1/75 = 0.013333
Variance: 1/75^2 = 0.0001777

Mean: 4% error
Variance: 10% error

The error is within reasonable bounds

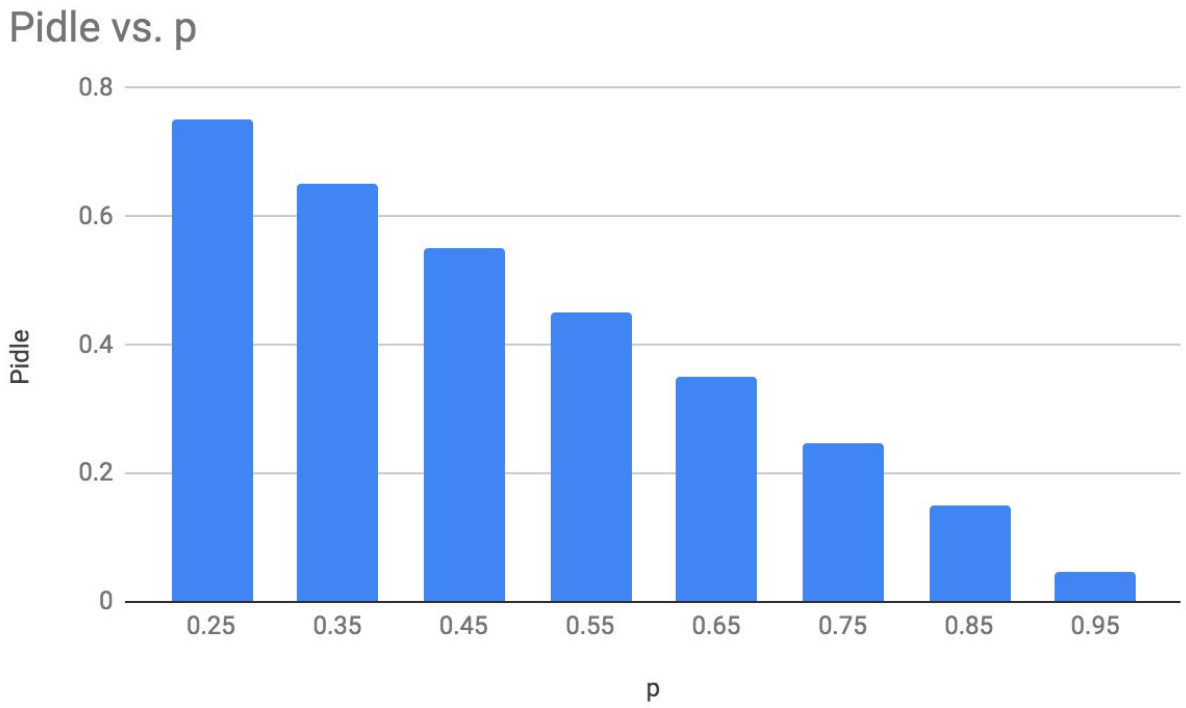
Question 3: M/M/1 Simulation Results

E[N] was calculated by finding the difference between number of packets that have arrived and number of packets that departed at each observer, and averaging the result.



E[N] appears to have a linear relationship with p. This can be explained by the fact that we are attempting to force more packets through the link at a time, resulting in more packets entering the queue.

Pidle was calculated by finding the number of observers that had an equal number of packets that have arrived and departed, and dividing by the number of observers.



As p increases, we can see that the amount of time the queue is idle decreases. The relationship appears to be close to linear.

Question 4: Testing M/M/1 with p = 1.2
E[N] = 107.286
Pidle = 0.0155767

We observe the same trend with E[N], where E[N] increases when p increases in size. We also observe the same trend with Pidle, where Pidle decreases as we increase p.

Question 5: Simulator Design

We used our simulator created in question 2 as the base for our finite buffer simulator, and mainly changes the departure calculation loop. We added the variables “j” and “bufferSize” to the main departure calculation loop. The variable “bufferSize” represents how many packets are in the buffer at a given state. The variable “j” points to the next packet that should be removed from the queue at a given state.

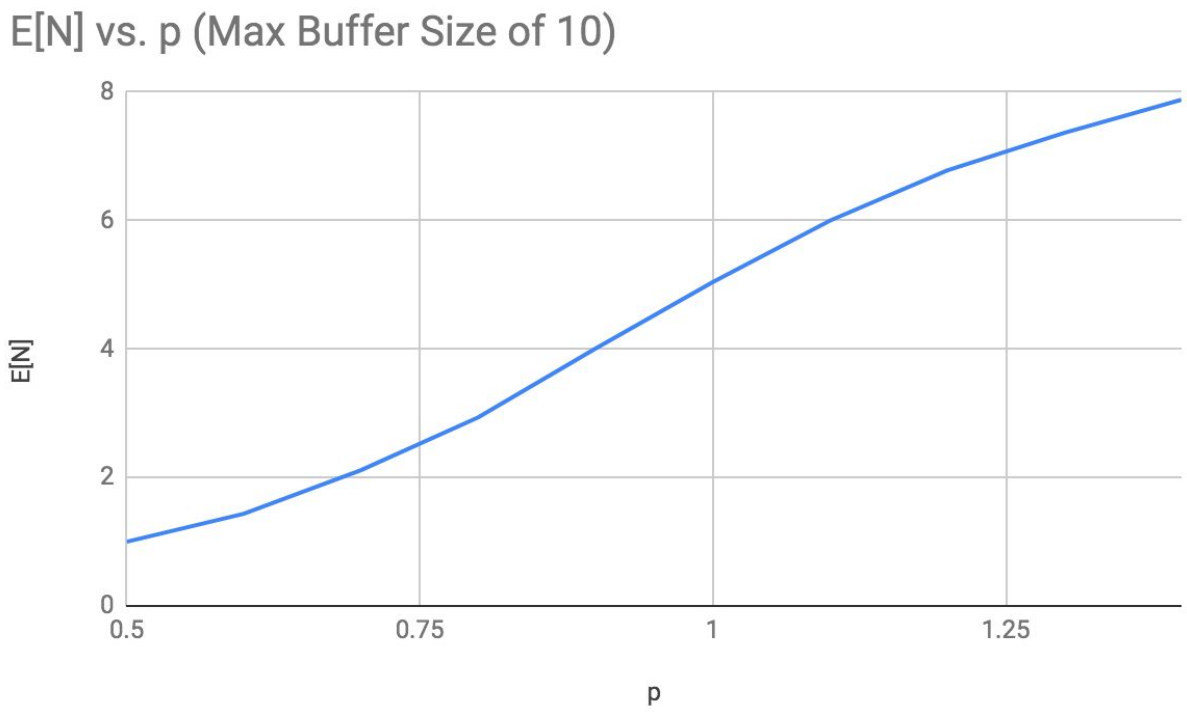
The main departure calculation loop runs through the list of packets sorted by arrival time. If the arrival time of the current packet is past the departure time of the previous packet, we can assume that the queue was empty. We can then calculate the departure time using the arrival time of the packet, set “bufferSize” to one, and have “j” point to this packet.

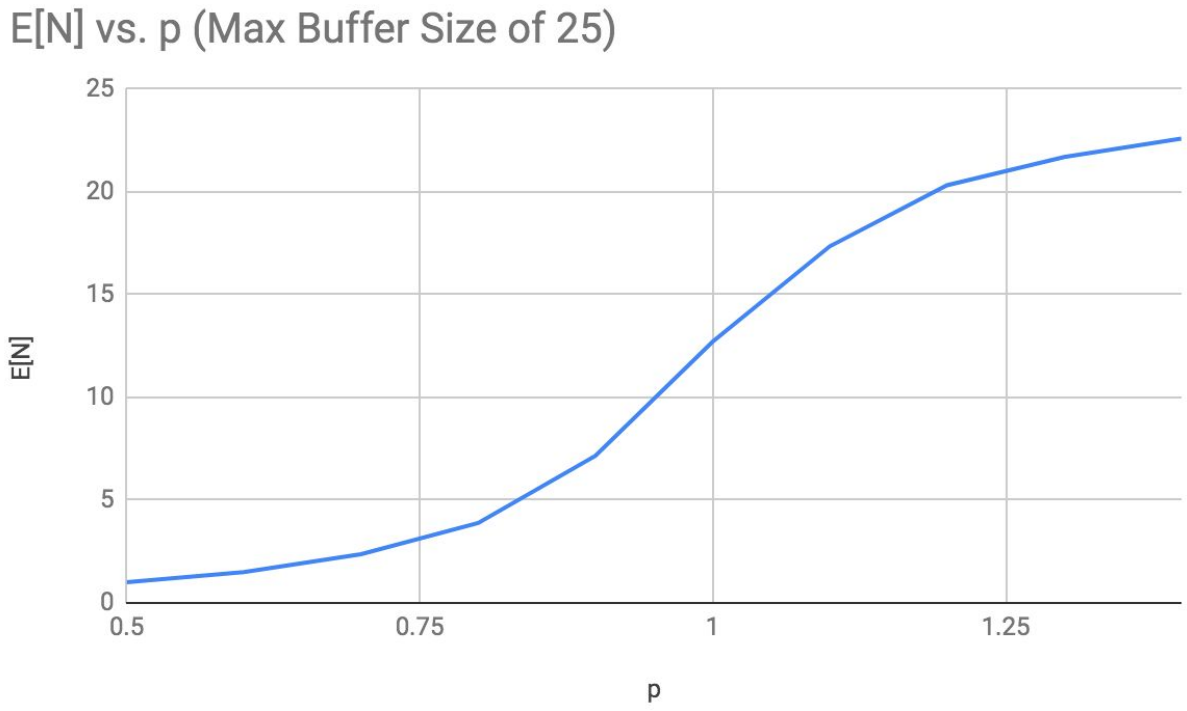
If the arrival time of the current packet occurs before the previous packet’s recorded departure time, we will attempt to add the packet to the queue. We will first check to see if the packet pointed to by “j” should leave the queue by comparing that packet’s departure time with the current packet’s arrival time. If the arrival time is after the other packet’s departure time, we will move “j” to the next non-dropped packet in the list and decrease the buffer size. This process will repeat until the packet pointed to by “j” has a departure time later than the current packet’s arrival time. After handling “j”, we will check to see if the queue can accept the current packet. If it can, we will calculate the packet’s departure time and add it to the queue, incrementing “bufferSize”. If we cannot, we will mark the packet as “dropped” and use the previous packet’s departure time as the current packet’s departure time.

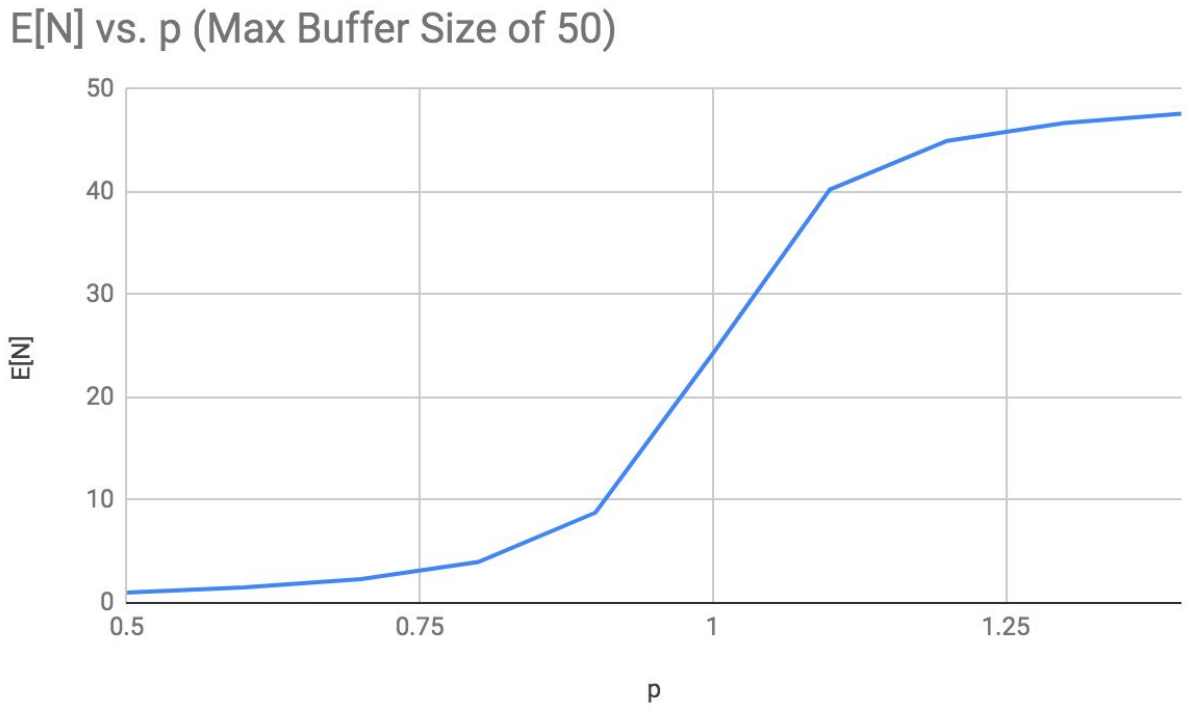
When generating observer events, we will keep a tally of dropped packets, but will not count these packets when calculating packets arrived and packets departed.

Question 6:

E[N] Graphs:



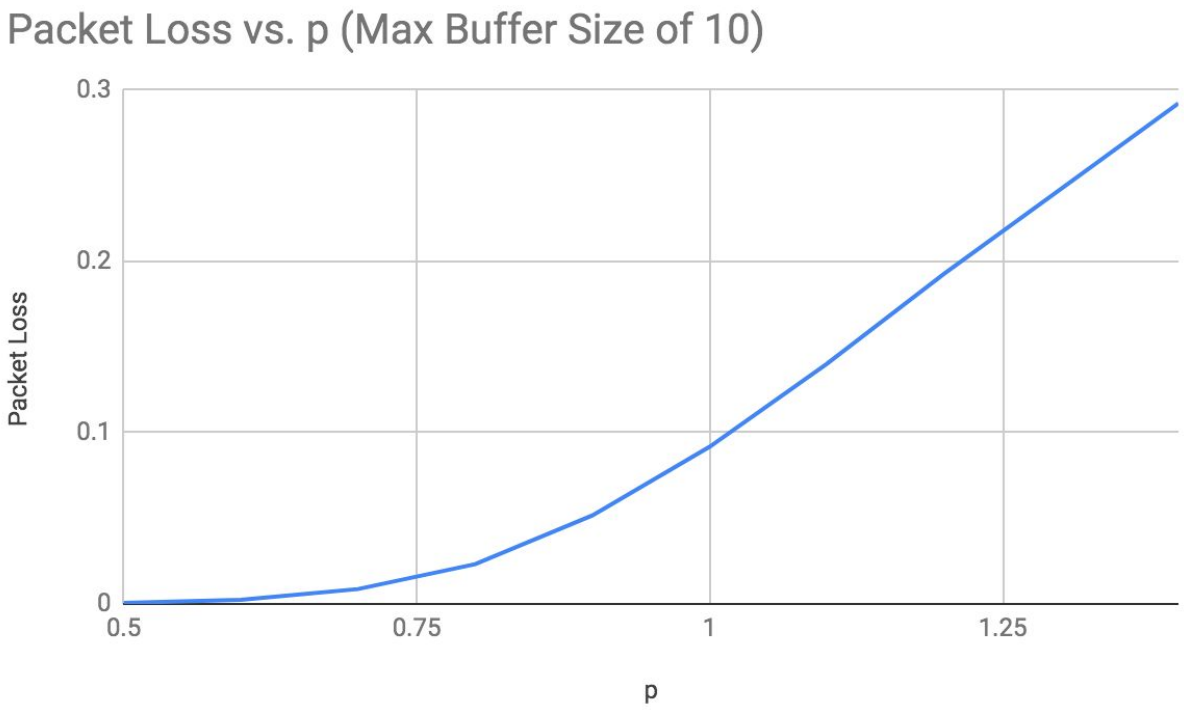


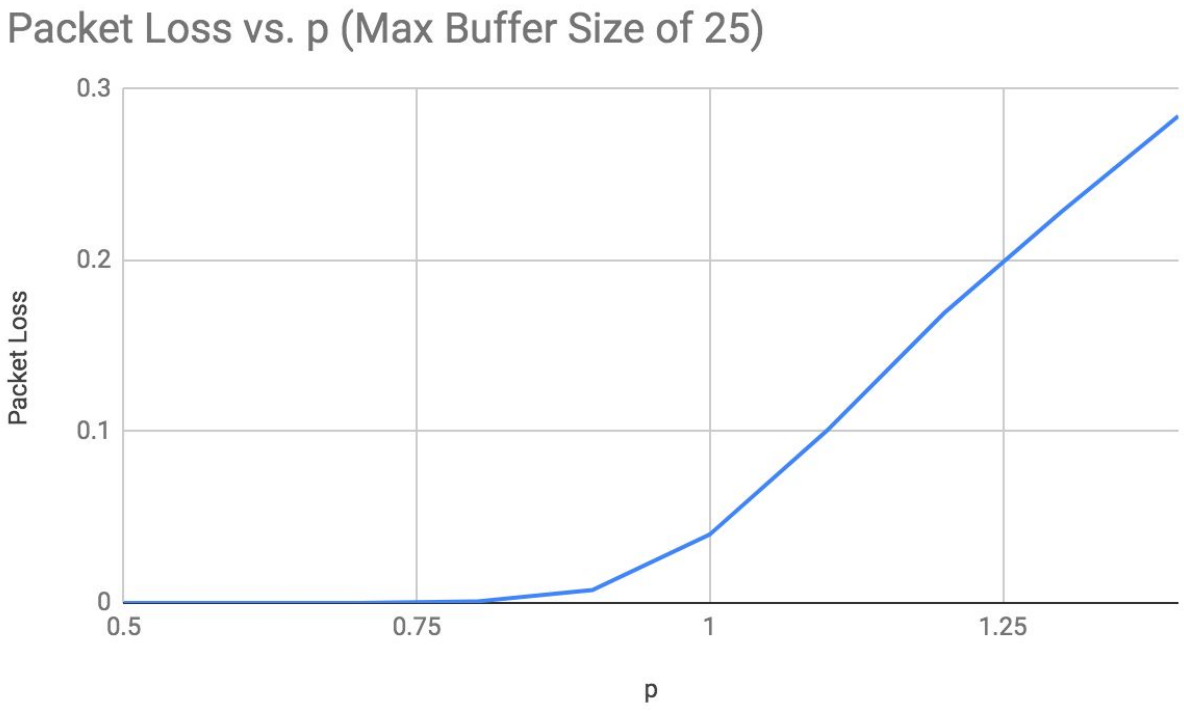


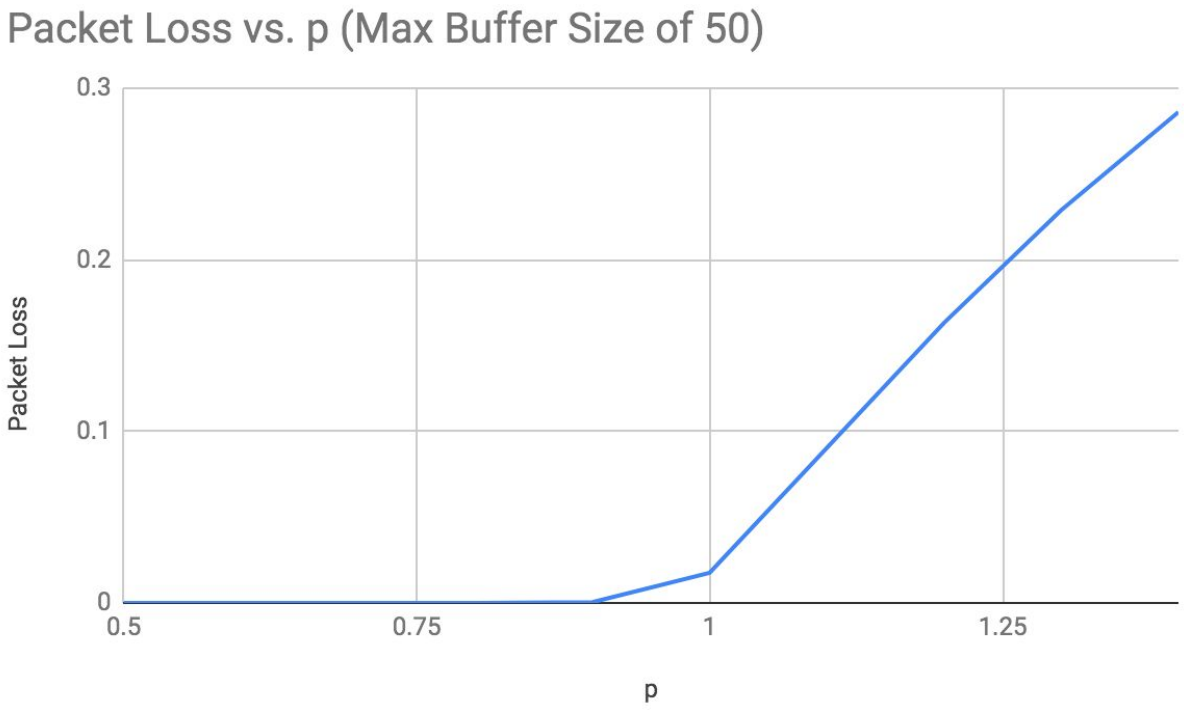
For the E[N] vs p graphs, we notice a couple of things. Firstly, for low values of p where the size of the buffer almost never exceeded, the curves are identical since the buffer size only matters when the buffer gets filled. We also notice that curves cross thought p=1 at exactly half the

buffer size. That is probably because at p=1 the expected value of the buffer size at any given point is half the buffer size. This is because at p = 1, the distribution of buffer size is even from 1-K. For p>1, we notice that E[N] asymptotes at K which makes sense since the buffer is constantly dropping some packets so it is full.

Packet Loss Graphs







We can see that for packet loss, as K increases, the graph will look closer and closer to a ramp function. This is because at p<1 as K increases, the chance of packet loss goes to 0 since, on

average, the server is processing packets faster then they come in. We also noted that the E[N] graph for k=10,25,50 looked similar for p<1. Since we know E[N] at any given P is on a exponential distribution, increasing K will make packet loss less and less likely as we are basically moving a boundary to the right on the exponential distribution. For p>1 we can see from the graph that packet loss increases linearly as P grows.