ORIE 4580: Simulation Modeling and Analysis

Final Project Report

December 2018

Instructor: Sid Banerjee

Alexa Durso, Zoe McCormick, Barbara Sudol, Max Wulff

Executive Summary

Date: December 2018
To: Llenroc Project Managers
From: Llenroc Simulation Team
Subject: Report of Simulation of BigRedCoin System

As a part of the simulation team at Llenroc University, our goal is to determine the feasibility of creating a virtual currency BigRedCoins (BRCs) and an online ledger that is a reliable marketplace for Llenroc's current students. To do this, we have created a simulation that captures the major aspects and parameters of the BRC system. Our main objectives for the simulation is to model the flow of transactions through the system in order to observe throughput, time spent in the system, delays in the system, as well as the rate of fees collected. Collecting this data will allow us to determine the optimal conditions for the given parameters of our simulator. This optimal solution allows the minimum amount of work per miner while minimizing the average delay and queue length in the system, which also proves the feasibility of the BRC system we have created and helps us understand the economics behind the BRC system.

To correctly simulate the parameters of the BRC system and verify our model's accuracy, we defined a set of assumptions to observe in our steady state metrics. Our assumptions behind the model include independent and exponential (with mean λ) arrival of all transactions, a constant number of miners in the system (N) with the same processing power, and uniform random fee rates and amounts. These assumptions allow us to have constant arrival and posting rates which greatly simplifies the model. In order to verify our simulation, we predicted and observed that a transaction with a higher BRC fee amount will spend less time in the system as the miners are more likely to include this transaction in their next block. Additionally, we compared the theoretical average number of transactions in the system to the simulated number. Lastly, we considered the behaviour of the model with different relative rates of arrival and processing time of transactions. As expected, the queue length remained stable and small when arrival < processing, and grew unbounded when arrival > processing. This further shows that our model functions properly and follows the queuing principles that guide our assumptions.

In order to recognize the optimal parameters for the BRC system, we used a variety of different simulation tests, and alternated which parameters were constant with which values we we tested for. Given an arrival rate of λ, adjustable mining rate of μ and number of transactions allowed per block K, our first approach was to arbitrarily fix parameters $\lambda = 120$ and $K = 120$, and test for the average delay and queue length for a range of values of μ. By varying μ with the given parameters, we were able to decrease our range for μ to [5,30] and found that the average profit per miner does not follow a specific trend. Next, we varied μ for varied K and $\lambda = 120$, and determined that K must be a value greater than 24. For $K = 25$, there was a significantly higher delay and queue length but only in the μ range [5, 10]. Additionally, for $K > 50$, all of the results are similar. Going forward, because a higher value of K did not affect the optimality of the system from our definition, we set $K=50$ for our block size for optimality.

We then discovered that while keeping $\lambda = 120$ constant, the optimal values for our system were $K \geq 50$ and $10 \leq \mu \leq 20$. We observed that a higher work rate corresponds to a shorter delay and queue length. From these results, we are able to conclude that the delay drops by about 0.01 time units on average for each unit of μ ($10 \leq \mu \leq 20$) and the queue length drops by about 1.2 transactions. Additionally, we concluded that the profit per miner still did not follow a visible trend due to randomly chosen fee values. Even with a fixed fee we still found that profit per miner was not affected by μ. Lastly, we wanted to compare the differences between the two endpoints for our optimal μ values. The largest difference between these two values is the effects on delay in system by fee amount. Our general recommendation for a given λ is to use $1/12*\lambda \leq \mu \leq 1/6*\lambda$, and to set K to a value above λ/μ, depending on how one wants profits distributed.

Our last objective is to understand the economics behind the BRC system once launched. One of our main concerns is that users expect that transactions are processed in a reasonable amount of time. We kept track of the ratio of the fee amount/fee rate for transactions, and computed an approximate relationship of value/rate ratio to delay times in the system, which ensures that transactions can be processed in a specific time period. Another concern is miner profitability - we calculated the maximum number of miners for which mining is still profitable, given that they pay a fixed cost per hour to mine. We observed that as the number of miners increases, the average profit per miner decreases. The breakeven point using our parameters including a mining price per hour of 0.5 is at around $N = 50$ miners, after which mining is no longer profitable. After considering these challenges which could affect the functionality of the system, we can affirm that our simulator is a functional and practical approximation.

Problem Description

Llenroc University is currently considering creating a virtual currency for its students. This marketplace where students can purchase and trade goods or services will use a cryptocurrency called BigRedCoins (BRCs). Some of the main concerns when creating the BRC system are guaranteeing minimal delays in transaction processing time, ensuring security without a central authority and validating the proper transaction amount between participants.

The goal is for BRCs to be a currency that can verify transactions, and ensure that: students can transfer money to each other, proper amounts are in accounts before the transfer, verification is done by individuals in the system, and fees are paid into the system with each transaction. The blockchain, or online ledger for the system, will keep track of all transactions between students. Some of its main features will include unique ids for BRC users, transactions that transfer between user accounts, and a BRC_core account with an initial amount of BRCs.

Transactions will be one of the most important aspects of the BRC blockchain. Transactions will compose of a sender id, receiver id, transfer amount, and transaction fees. For the transaction to be added to the blockchain, participants must submit a transaction request, the request must be verified by miners, the transaction must be blocked together with other transactions, and finally the transfer occurs when the block with the transaction is added to the end of the blockchain.

Lastly, we will explore the relationships between the parameters and metrics of interest in our blockchain by simulating it as a queue. We will determine the optimal number of active miners, transactions per block, block mining rate, transaction fee amount, and transaction arrival rate.

Modeling Approach and Assumptions and Verification

Our aim is to simulate the flow of transactions through the BRC system in order to collect data such as throughput, time spent in system per transaction and fee collection. Given that our simulation is designed with transactions at the forefront, there are many assumptions made in order to eliminate the need to keep track of each individual BRC user and miner's activity. First, we assume the arrival of all transactions are independent and arrive in an exponential distribution (poisson process) with an adjustable parameter λ.

Another major assumption corresponds to the behaviour of the miners: over the life of the simulation, the number of miners in the system (denoted as N) remains constant and these N miners have the same processing power. The miners are all assumed to have independent interest in earning the maximum amount of fees so, as soon as a block is posted, all miners simultaneously will add the K highest transactions to their new blocks and immediately begin working (K defined as the maximum fixed number of transactions per block). These assumptions allow us to turn block posting into an exponential random variable whose parameter should depend on an 'exponential race' between the N miners (See Appendix A). However, in order to simplify further, we assume that the rate of mining μ remains constant and independent of the number of miners, as the rate can also be controlled by adjusting the difficulty of the cryptographic puzzles solved. Because of these assumptions we also no longer have to keep track of individual miners and just consider how the constant rate in and rate out affect a queue of transactions.

Another assumption important to the structure of our model relates to the distribution of fees and transaction amounts. Each transaction is its own instance of an object which arrives exponentially, waits in a transaction queue, is added to a block and eventually is posted. The transactions object has two attributes: fee rate and fee amount, which we assume are random (50% chance of 1% or 2% fee and a uniform random transaction value between 5 and 25). The assumption of the distribution of fee amount is important because it further allows us to consolidate all individuals into a single, central poisson process of transaction arrivals.

Since we consolidate all individual ledgers and block postings into simple exponential variables, we are able to apply concepts from queueing theory to this problem. Given our system with a queue of transactions (including a predefined block of the most lucrative K transactions), we define two distinct events that can move transactions into or out of the system: the event that a transaction is posted by a user and arrives to the queue, or the event that a miner solves the cryptographic puzzle and posts a block, and the predefined min(K, queue size) transactions are taken out of the queue (see Appendix B). The parameters which specify rate of mining and rate of transaction arrival are assumed constant throughout the life of the simulation, and the range of values that we test the simulation over are specified below in the *Model Analysis* section. Let us note that the main technical constraint imposed by the cryptographic protocols is that μ can be at most 30 blocks/hr, but K and λ are technically unbounded.

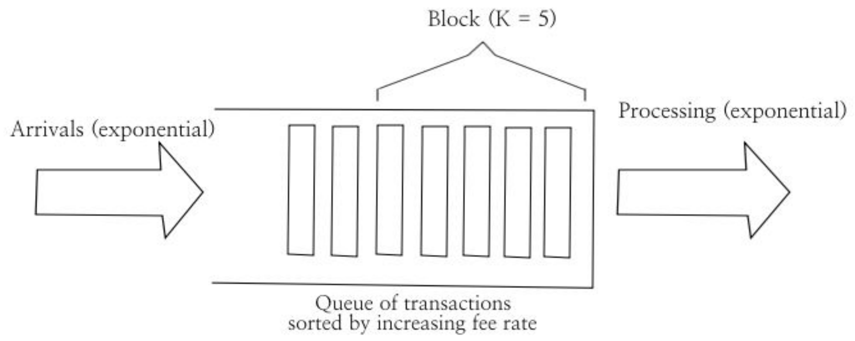
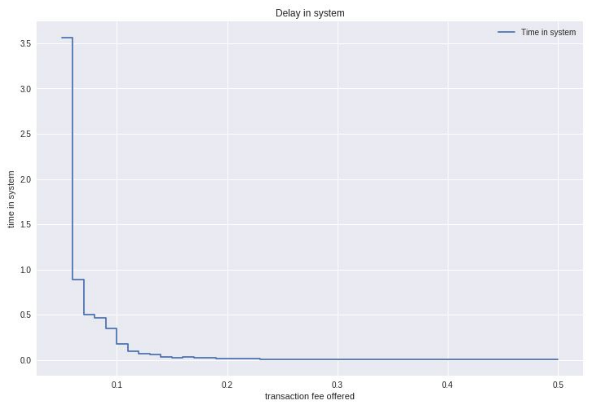


Figure 1: A simple diagram illustrating our model

Technical specifications of our model are located in Appendix B and the source code of the model is located in Appendix D.

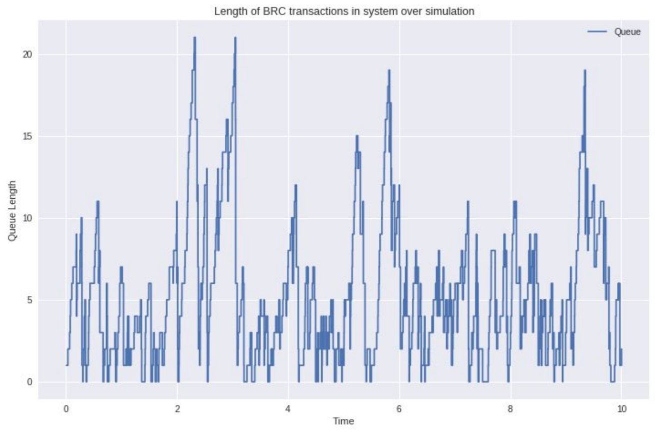
In order to verify our model's accuracy, we observed some steady state metrics as well as trends in our simulation and compared them to their theoretical values. First, we checked the relationship between a transaction fee and its delay. We expect this to be a decreasing trend - meaning the higher the fee (in BRC) offered to the miner, the less time it should spend in the system since miners are more likely to include it in the next block being posted. We saw the following output (and the same for a variety of input parameters), which validates our claim:

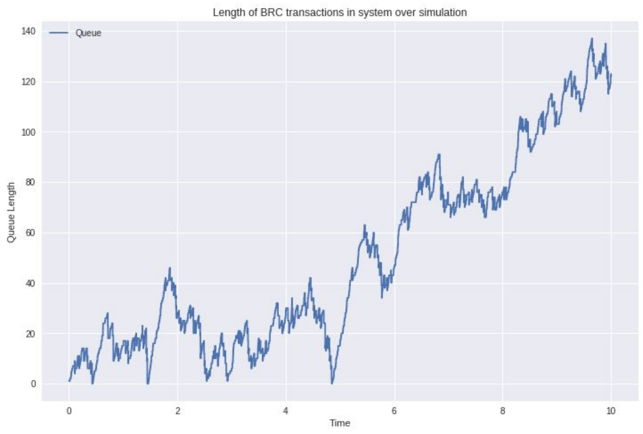


We also used queueing theory to calculate expected average number of transactions in system (See Appendix C) and compared this to our simulation output. The results were:

Expected = 1.73 transactions; Actual = 1.65 transactions

Lastly, we considered the relative rates of arrival and processing in the system, and observed the trends when $\lambda > K*\mu$ and $\lambda < K*\mu$ (details of the applicable queueing physics principle can be found in Appendix C). We expect the queue size to be relatively small or contained when $\lambda < K*\mu$, and large or unbounded when $\lambda > K*\mu$. The results of our model in this regard:





Queue length: $\lambda = 50$, K = 5, $\mu = 20$

Queue length: $\lambda = 110$, K = 5, $\mu = 20$

Clearly we can see the queue becomes unbounded when our utilization condition is broken as predicted, which suggests that our model functions as the problem specified.

Model Analysis

Key: λ = Transaction Arrival Rate, μ = Block Posting Rate, K = Transactions Per Block,

N = Number of Miners

To analyze our model, we first decided to define optimizing the solution as minimizing the amount of work per miner (μ) while also minimizing the average delay and queue length in the system. The layout of our analysis is as follows:

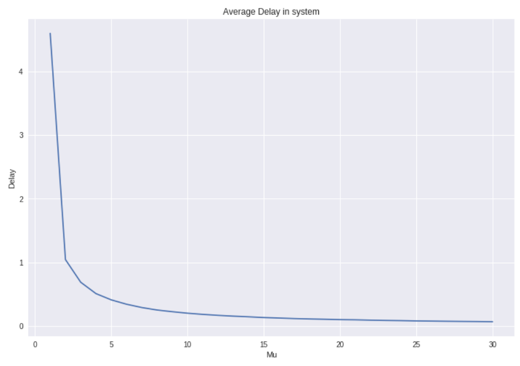
1. Test 1: Varying μ
   a. Graph 1.a: Models the average delay of transactions in Test 1.
   b. Graph 1.b: Models the average queue length of transactions in Test 1.
   c. Graph 1.c: Models the average profit per miner in Test 1.
   d. Graph 1.d: Models the average throughput of transactions in Test 1.
   e. Graph 1.e: Models the average total fees collected in Test 1.
2. Test 2: Varying μ for Varied K
   a. Graph 2.a: Models the average delay of transactions in Test 2.
   b. Graph 2.b: Models the average queue length of transactions in Test 2.
   c. Graph 2.c: Models the average profit per miner in Test 2.
   d. Graph 2.d: Models the average throughput of transactions in Test 2.
   e. Graph 2.e: Models the average total fees collected in Test 2.
3. Test 3: Varying μ Over Optimal Range
   a. Graph 3.a: Models the average delay of transactions.
   b. Graph 3.b: Models the average queue length of transactions.
   c. Graph 3.c: Models average profit per miner - comparing a fixed fee of .01 vs .02.
4. Performance Metrics for Optimal Parameters.
   a. Graph 4.a: Queue length over time of simulation.
   b. Graph 4.b: Throughput of simulation.
   c. Graph 4.c: Average delay as a function of fee amount.
   d. Graph 4.d: Rate of fees as a function of time.
5. Economics Considerations
   a. Graph 5.a: Average time in system over ratio of transaction amount to fee rate.
   b. Graph 5.b: Average profit per miner with mining price *c* as a function of N.
6. General Recommendations
   a. Graph 6.a: Average delay for transactions for varying values of λ.

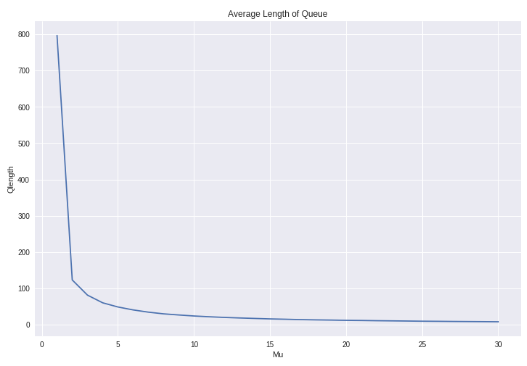
Test 1: Varying μ

We fix λ = 120 as suggested. In order to keep K*μ > λ and vary μ over its entire range, we fix K at its maximum value = 120 (when μ = 1, K = 120 in order to keep a convergent system). In the following graphs, we look at the average delay in the system, the average length of the queue, and the average profit per miner (not including any per hour or entry fee for miners). To gather this data, we run the simulation 20 times for each value of μ in the range [1, 30], and average the results for delay, queue length, and profit.

Parameters: λ = 120, K = 120, N = 20, t_end = 100, μ=[1,30]

Results:

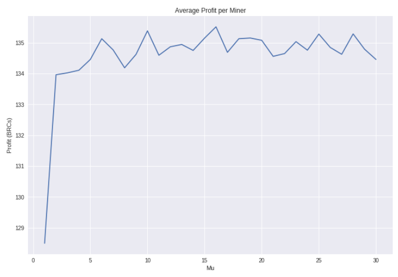


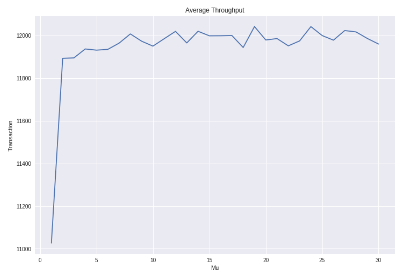


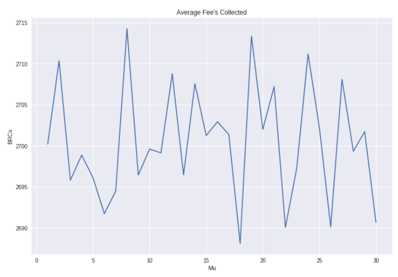
Graph 1.a: Average Delay of Transactions for μ in range [1, 30].

Graph 1.b: Average Length of Queue for μ in range [1, 30].

We can see here that, predictably, as μ increases in the range [1, 30], the average delay of transactions in the system decreases. Following this trend, the average length of the queue per value of μ appears to decrease with the same distribution. This is unsurprising as queue length and delay time should be directly correlated.







Graph 1.c: Average Profit per Miner for μ in range [1, 30]

Graph 1.d: Average Throughput of Transactions for μ in range [1, 30]

Graph 1.e: Average Total Fees Collected for μ in range [1, 30]

It appears from the data collected here that, as μ increases, the average profit per miner increases sharply for μ = [1,5] and then steadies around 135 BRCs/miner for the rest of our range of μ. This is also expected considering the fee rate is randomly chosen. The average throughput of the system seems to be distributed in the same way, and the average of fees collected does not seem to be dependent on the value of μ at all.

The conclusions from this first set of data lead us to believe that a μ value in the range [5, 30] will give similarly optimal results. In any case, a higher μ generates a more optimal system, so the chosen μ value should be as high as possible in context, but in order to find a minimal μ as we defined an optimal solution, we will choose μ in range [5, 30] based on these results. For the next section, we will test μ in the range [5,30] and vary K in order to analyze how this value affects the system.

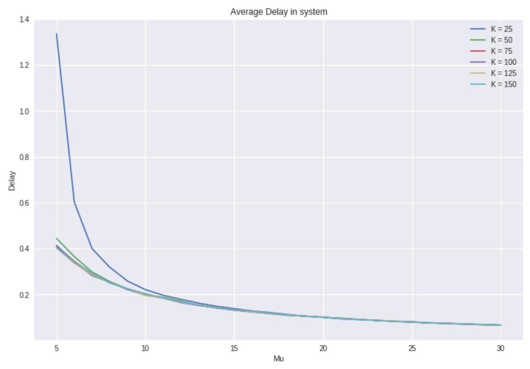
Test 2: Varying μ for Varied K

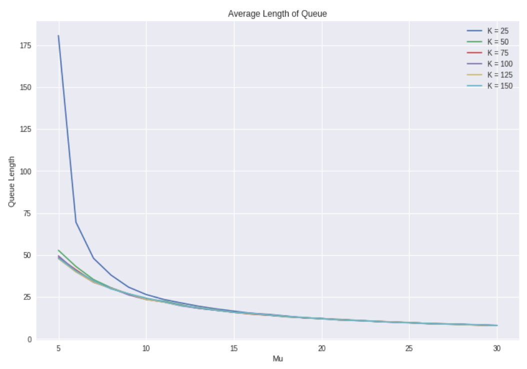
Using what we observed to be an optimal range of μ, we next test our model for different values of K. For this part of testing, we will test μ in range [5, 30] so therefore K must be at least 24, as we are testing μ in range [5, 30] and (5*24 = 120). We leave N and λ fixed at the same values.

Parameters: λ = 120, N = 20, t_end = 100, μ = [5,30].

K = [25, 50, 75, 100, 125, 150]

Results:

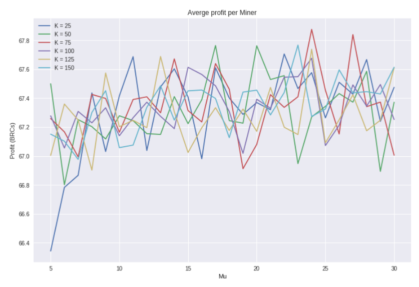


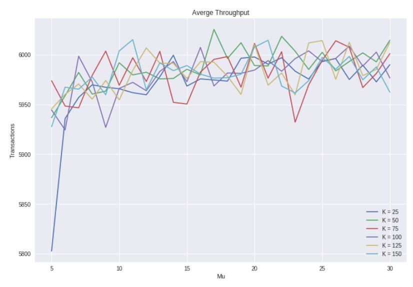


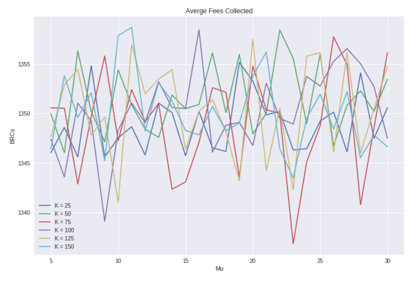
Graph 2.a: Average Delay of Transactions for μ in range [5, 30], varying K.

Graph 2.b: Average Length of Queue for μ in range [5, 30], varying K.

We can see from these results that the the average delay of transactions in the system and the average queue length converge to nearly identical results for values of μ > 10. The K = 25 run shows significantly higher delay and queue length but only in the range [5, 10]. For K > 50, all of the results are similar. Because a higher value of K does not have an effect on the optimality of the system from our definition, setting K=50 is a reasonable block size for optimality.







Graph 2.c: Average Length of Queue for μ in range [5, 30], varying K.

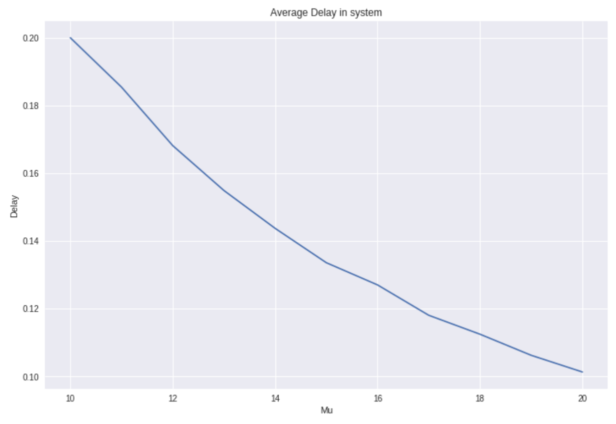
Graph 2.d: Average Throughput of Transactions for μ in range [5, 30], varying K.

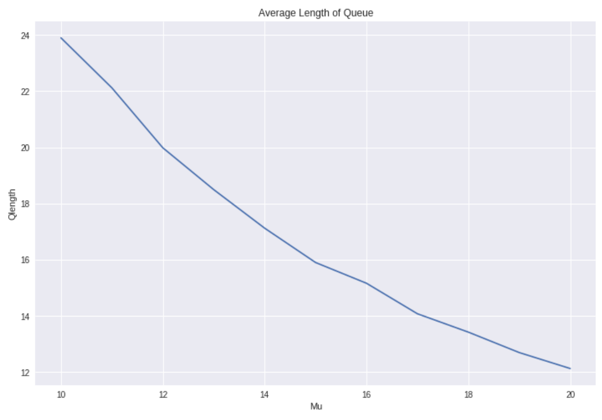
Graph 2.e: Average Total Fees Collected for μ in range [5, 30], varying K.

Seeing from the above three plots, there is no discernible trend for average profit per miner, average throughput, or total fees collected with varying K.

Test 3: Varying μ Over Optimal Range

Based on the above analysis, we can define a good enough range for optimality based on our definition of a μ value in [10,20], K = 50 and λ constant. This optimal parameter set gives the following solutions over the range μ = [10,20]:



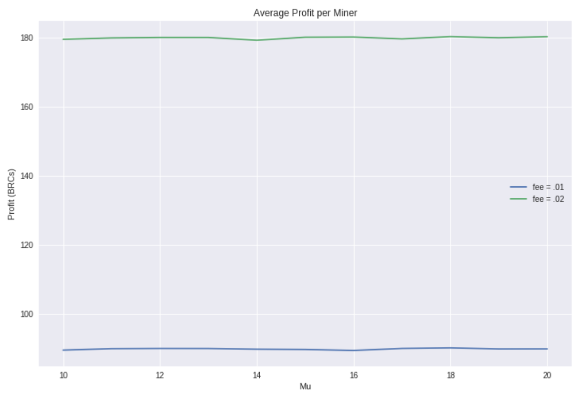


Graph 3.a: Average Delay of Transactions over μ in range [10, 20].

Graph 3.b: Average Queue Length for μ in range [10, 20].

The observations here make sense, as obviously a higher work rate corresponds to a shorter delay and length of queue. Based on these results, we can see that the delay drops by about 0.01 hours on average for each unit of μ, 10 <= μ <= 20, and the queue length drops by about 1.2.

We expect profit per miner to not follow a visible trend again because the fee value is randomly chosen in our model. In order to visualize the profit per miner as a function of μ, the following graph shows what the system looks like for a fixed fee rate. Clearly, the higher fee value will correspond to a higher profit per miner as the number of miners is fixed.

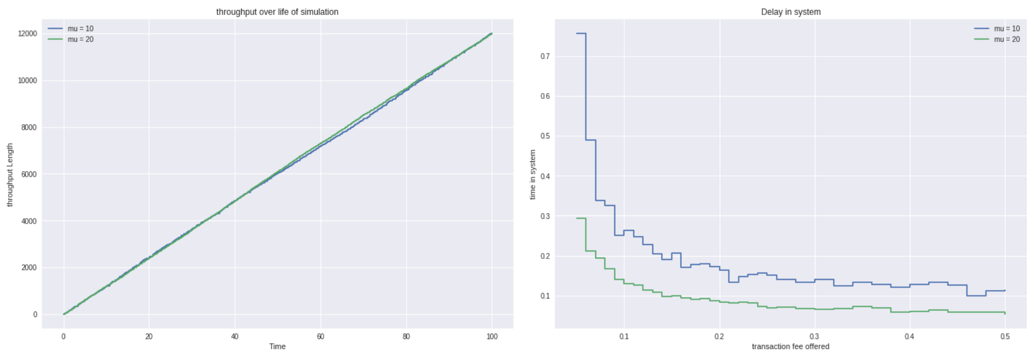


Graph 3.c: Average profit per miner for μ in range [10, 20], fee = .01 or .02.

Based on this, we can officially conclude that the profit per miner does not vary based on μ.

Test 4: Performance Metrics for Optimal Parameters

Comparing the two endpoints of our range for μ, we can see the difference in the performance metrics for μ = 10 (blue) and μ = 20 (green):



Graph 4.a: Throughput over time, μ = 10, 20.

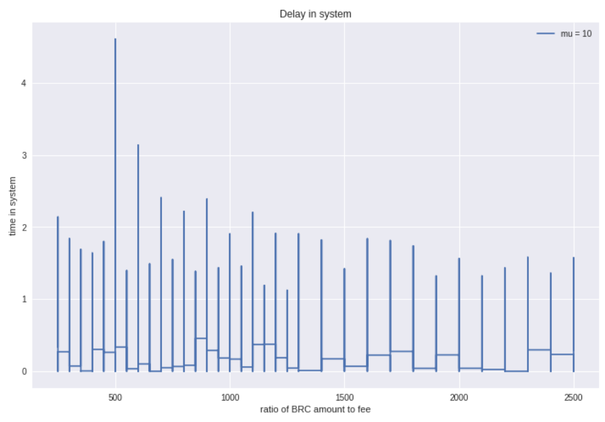
Graph 4.b: Time in system as a function of transaction fee, μ = 10, 20.

μ	average time in system	total amount collected in fees	total throughput (transactions)	total throughput (BRCs)
10	0.218630717	2662.62	11830	177431
20	0.097640403	2712.45	12090	180899

The most striking differences in results between when μ = 10 (blue) and μ = 20 (green) are the effects on the delay in system by fee amount, and the metrics showing the difference in average time in system. This correlates to graphs 3.a. and 3.b. showing the delay and queue length for the range of μ from 10 to 20. Based on this, recommendations for an optimal system would depend on the possible values of μ and the penalty cost for a higher μ.

Test 5: Economics

a. The first question to consider is how to ensure that transactions will be processed in specific amounts of time. To test this, we kept track of the ratio of the transaction amount/fee rate for transactions. We then modeled the time in the system for each transaction (delay time) by its ratio. The results are as follows:



Graph 5.a: Average time in system versus ratio of transaction value:fee rate.

This was modeled using parameters μ = 10, K = 50, N = 20, λ = 120, and t_end = 10000. Using this, we can see that as the ratio increases, the time in system decreases. Using this, one could calculate the fee rate they should use based on the value of their transaction.

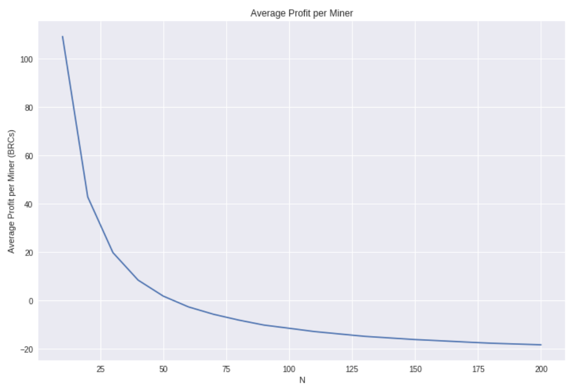
For example, say a user wanting to transfer 20 BRC's needs their transaction posted in 2 hours. Approximately, they could look at the above graph and note that no transactions with a ratio greater than 1500 were in delay longer than 2 hours. Using this, the ratio they're looking for is value/fee rate = 1500. They can then plug in their transaction value and calculate:

$$20/\text{feeRate} = 1500 \Rightarrow \text{feeRate} = 0.013333$$

This is not an exact solution, but gives an approximate solution of what value/rate ratio on transactions corresponds to certain delay times in the system.

b. The second economics question to consider is, if there's some fixed cost of *c* BRC's that miners pay per hour for mining, how many active miners can be in the system for mining to be profitable?

To model this, we again use μ = 10, K = 50, N = 20, λ = 120, and t_end = 10000. The average profit per miner will be the BRC's made per miner from mining minus *c**(total hours spent mining). We model this profit for different values of N, using an arbitrary *c* = .5 BRC's.

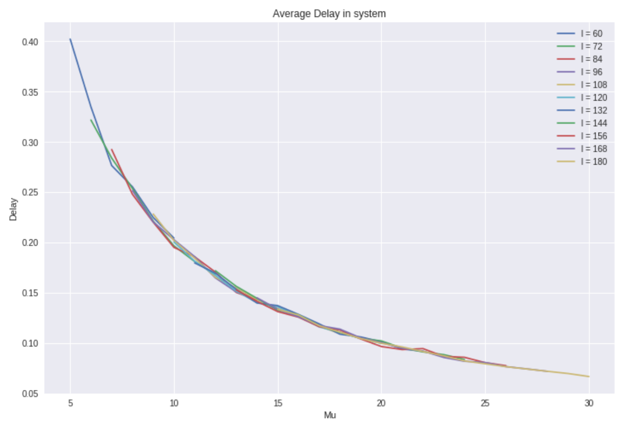


Graph 5.b: Average profit per miner (BRC's) with mining price *c* BRCs/hr for varying N

It's clear here that as the number of miners increases, the average profit per miner decreases. The breakeven point using our parameters and a mining price per hour of *c* = 0.5 is at around N = 50 miners. After this point, mining is no longer profitable.

General Recommendations

Based on our above analysis, we tested to determine an optimal range μ based on a more general, varying λ. Previously, we kept a fixed arrival rate of λ = 120, but to apply to a more realistic position in this problem, we need to consider an unknown transaction arrival rate. In the following graph, we model for varying values of λ the average delay of transactions in the system as a function of μ = [1/12 λ, 1/6 λ]:



Graph 6.a: Average delay of transactions as a function of μ, for varying λ.

Graph 6.a. shows that the optimal range for μ that we extrapolated from our above analysis applies generally to any given value of λ, as it shows that the average delay of the system modeled over different values of λ converges when μ = [1/12 λ, 1/6 λ]. Therefore, our general recommendations are, for any given transaction arrival rate λ, use μ in range 1/12*λ to ⅙*λ, and choose a K value such that K > λ/μ.

Conclusions

Key: λ = Transaction Arrival Rate, μ = Block Posting Rate, K = Transactions/Block

To briefly summarize the results from our analysis:

- An optimal solution is one which minimizes the work for the miners while keeping transaction times low, profits for the miner high, and the transaction queue length small.
- The optimal rate of μ to keep the queue length and transaction time down is somewhere between 1/12 and ⅙ of λ, given all other parameters are fixed.
- Changing K does not have a significant effect on on delay, queue size, or profit per miner given that K is greater than λ/μ. However it does change how the profit is distributed. If K is high then a few miners will make a large profit. If K is small then more miners will make smaller profits.
- Over the range of the optimal μ the profit per miner does not change, given other parameters do not change.
- Our recommendation is to set μ to between 1/12 and ⅙ of λ. Set K to some value above λ/μ, depending on how you want profits distributed.

Appendices

Appendix A: Exponential Race

For exponential random variables $X_{1,}$ $X_{2,}$ $X_{3\ldots}$ with respective means $\mu_{1,}$ μ_2 μ_3: the random variable defined as $\min(X_{1,} X_{2,} X_3)$ is also distributed exponentially with mean $= \mu_1 + \mu_2 + \mu_{3.}$ Therefore, the event of the first miner finishing and posting a block can be modeled as a random variable with mean $(\mu_1 + \mu_2 + \ldots) = (N{*}\mu)$. However, note that we are assuming that we will control μ to be independent of N.

Appendix B: Technical Model Specifications

- Our transaction queue is an object, a main attribute being an array of transactions.
  - The transactions are also objects, with attributes fee amount and fee rate, specified by the outcome of a random variable when it is created.
  - The array of transactions is sorted by increasingly by transactions fee attribute, which is calculated as fee rate * fee amount.
  - The transaction queue also has methods to finish a block, where it removes the first n transactions from the queue, based on a passed in parameter n.
  - The number of transactions that was left in the queue at the time of the last posting was saved as K, and when the event of a block posting is generated next, this value K is passed into the finish block method, and those K leave the queue. Then, the minimum(K, length of queue) becomes the new value K.
  - We also have a method to add a transaction to the queue, and re-sort the array to ensure it is still in fee decreasing order. The last helper method we implemented in the queue object returns the number of transactions in the system at the time it is called, which helps with collecting data.
- The event generator is based on an exponential random variable for time, with mean = $1/(\mu+\lambda)$, similarly to the exponential race described in *Appendix A*. When a new event time occurs according to this random variable, we decide which event it is (arrival to queue or block posting) based on a uniform random variable, with probabilities of each event proportional to $\lambda/(\lambda+\mu)$ or $\mu/(\mu+\lambda)$.

- For analysis, we append to variables such as time, queue length, fee collected, number of transactions posted per block, etc to arrays at each iteration.

Appendix C: Queueing Theory Math for Verification

For a M/M/C queue, the expected long run average number of entities or 'customers' in the system is equal to:

$\rho / 1 - \rho * C(c, \lambda, \mu) + c*\rho$

Where $\rho = \lambda/K*\mu$, $c = N$, and C is the Erlang equation:

$$C(c, \lambda/\mu) = \frac{\left(\frac{(c\rho)^c}{c!}\right)\left(\frac{1}{1-\rho}\right)}{\sum_{k=0}^{c-1} \frac{(c\rho)^k}{k!} + \left(\frac{(c\rho)^c}{c!}\right)\left(\frac{1}{1-\rho}\right)} = \frac{1}{1 + (1-\rho)\left(\frac{c!}{(c\rho)^c}\right)\sum_{k=0}^{c-1} \frac{(c\rho)^k}{k!}}$$

We essentially thought of our setup as a M/M/N queue where there are c = K servers, since each server post K blocks at each completion of their service, and based our calculations off of this.

We fixed $\lambda = 50$, $\mu = 30$ and $K = 4$ as reasonable but random parameters in order to directly compare the theoretical answer and the simulated answer with the same inputs.

Queueing physics principle: For queue to be stable, average 'work' coming to the system must be less than the average work capacity of system, meaning λ must be less than $K*\mu$, so $\rho < 1$ for the queue to stay bounded over long periods of time.

Appendix D: Original Queue Simulation

```
import numpy as np
import scipy.stats as sc
import matplotlib.pyplot as plt
%matplotlib inline
import pandas as pd
np.random.seed(0)
import math

class Queue(object):
    name = ""
    capacity = 1
    serviceRate = 1
    waitingentities = []

    # Initialize a Queue Object
    def __init__(self, name, capacity, serviceRate, waiting, inSystem):
        self.name = name
        self.waitingentities = waiting
        self.capacity = capacity
        self.serviceRate = serviceRate
        self.inSystem = inSystem

    # Print a Queue Object
    def __repr__(self):
        queue = self.name + " queue: " + str(self.waitingentities)
        system = self.name + " system: " + str(self.inSystem)
        return queue + " \n " + system

    # A new Transaction arrives at this Queue
    def ArriveatQueue(self, newarrival):
        self.waitingentities.append(newarrival)
        self.waitingentities.sort(key = lambda x: x.fee, reverse = True)
```

```
     #Given number of transcations, remove this number from the top of the queue
     def FinishBlock(self, currentBlock):
         newentities = []
         for e in range(currentBlock,np.size(self.waitingentities)):
             newentities.append(self.waitingentities[e])
         self.waitingentities = newentities

     #Return the current size of the queue
     def inSys(self):
         return np.size(self.waitingentities)

     # Print an Transaction arrival time
     def __repr__(self):
         return str(self.starttime)

 # General class for the transactions that move through the Queues
 class Transaction(object):
     starttime = 0
     value = 0
     feerate = 0
     fee = 0
     # Initialize an Transaction
     def __init__(self, start, val, f):
         self.starttime = start
         self.value = val
         self.feerate = f
         self.fee = f*val
     # Print an Transaction (just arrival time)
     def __repr__(self):
         return str(self.starttime)
```

```
################################################################################
################################################################################

#Parameters
mu = 30 #block mining rate; upper bound = 30/hr
K = 3 #number of transactions per block
lam = 70 #transaction arrival rate; reccomended 120/hr
mineRate = mu #exponential race between all miners
totalRate = mineRate + lam
N = 10 #not used

#initialize our central queue
pendingTransactions = Queue("pendingTransactions", N, mineRate, [],[])

# Store data
times = []
transactionQlength = [] #record queue length of diagnosis
timeinSys = []  #record total time in system for patient
feeRate = [] #record the rate that was collected to average later
feeAmount = []
through = []
feeC = 0 #keep track of cumulative fee for one block
feeBlock = [] #keep track of fees for all blocks over time
overallCumulativeFees = 0
delayInfo = [] # keep track of delay info


#Initializations for the loop
currentBlock = 0
t_end = 5 #length of the simulation
t = 0 #start
throughput = 0
```

```
#Event generator
while (t < t_end):
  nextEventTime = t + np.random.exponential(1/totalRate)
  t = nextEventTime
  U = np.random.rand()
  if U <=mineRate/totalRate:
    event = 0 #a miner posts a block
  else:
    event = 1 #a transaction arrives

  # Do the events corresponding to whatever the "soonest" next time is

  if event == 0: # a miner posts a block, multiple transactions leave the queue
    #post the block
    for i in range(currentBlock):
      timeinSys.append(t-pendingTransactions.waitingentities[i].starttime)

      # delay in system
      delayInfo.append(pendingTransactions.waitingentities[i].fee)
      feeC += pendingTransactions.waitingentities[i].fee

    pendingTransactions.FinishBlock(currentBlock)
    #add whatever is about to be posted to the throughput
    throughput = throughput + currentBlock
    #get the new block to be posted next time
    currentBlock = min(pendingTransactions.inSys(), K)
    #store fee data
    feeBlock.append(feeC)
    overallCumulativeFees += feeC
    feeC = 0
```

```
  else: #arrival to the queue
    #add random valued transaction - between 5 and 25 - to the queue
    #get a random fee - uniform 1 or 2 percent
    arrivalVal = np.random.randint(5, 26)
    rate = 0
    #determine random fee rate
    U = np.random.rand()
    if(U <= 0.5):
      rate = 0.01
    else:
      rate = 0.02
    #make a new transaction instance to add to queue
    newArrival = Transaction(t, arrivalVal, rate)
    #add new transaction to queue
    pendingTransactions.ArriveatQueue(newArrival)
    #append data
    feeRate.append(rate)
    feeAmount.append(rate*arrivalVal)
    feeBlock.append(0)

  # Record data
  times.append(t)
  transactionQlength.append(np.size(pendingTransactions.waitingentities))
  through.append(throughput)

# Make an informative plot of queue length over time
plt.figure(figsize=(12,8))
plt.step(times, transactionQlength, label="Queue", where='post')
plt.ylabel("Queue Length")
plt.xlabel("Time")
plt.legend()
plt.title("Length of BRC transactions in system over simulation")
plt.show()
print(np.average(transactionQlength))
```

```
#Plot throughput over time
plt.figure(figsize=(12,8))
plt.step(times, through, label="throughput", where='post')
plt.ylabel("throughput Length")
plt.xlabel("Time")
plt.legend()
plt.title("throughput over life of simulation")
plt.show()


# delay in system
set_fees = list(set(delayInfo))
delay_list = {set_fees[i]: [] for i in range(len(set_fees))}
for i in range(len(timeinSys)):
  delay_list[delayInfo[i]].append(timeinSys[i])
avg_delay = {i: np.mean(delay_list[i]) for i in delay_list}
avg_delay = dict(sorted(avg_delay.items()))

delay_fees = [i for i in avg_delay]
delay_time = [avg_delay[i] for i in avg_delay]

#plot time spent in system
plt.figure(figsize=(12,8))
plt.step(delay_fees, delay_time, label="Time in system", where='post')
plt.ylabel("time in system")
plt.xlabel("transaction fee offered")
plt.legend()
plt.title("Delay in system")
plt.show()
```

```
#plot fees paid per event
plt.figure(figsize=(12,8))
plt.step(times, feeBlock, label="Fees paid non cumulative", where='post')
plt.ylabel("Fee in BRC")
plt.xlabel("time")
plt.legend()
plt.title("Fees paid non cumulative")
plt.show()

#print more metrics
print("Overall Cumulative Fees Paid over time = BRC's entering the system:", overallCumulativeFees)
print("Fees Paid into the system per hour=", overallCumulativeFees/t_end)
print("Average time in system", np.average(delay_time))
print("Average transaction amt", np.average(feeAmount))
```