REPORT FOR THE BRC EXECUTIVE TEAM

Jiacheng Han, Andy Wu, Jacob Wyrick, Sidney Yune

EXECUTIVE SUMMARY

In this project we were tasked with creating a discrete event simulator to capture some insights about the Big Red Coin (BRC) system, mainly how to best configure the system to minimize delays. The main system parameters were the block mining rate μ, the number of transactions per block K, and the transaction arrival rate λ and these were allowed to vary to get a better idea of the system sensitivity to these parameters. In order to gain any useful data about the steady state performance of the system the parameters must be adjusted to be able to handle all incoming transactions without a large backlog accumulating. To avoid this happening the parameters should be set such that $\mu K > \lambda$. To observe the effect of these parameters on system performance, data was collected on three basic metrics for the system: (1) throughput of system in terms of number of transactions and BRCs, (2) delay experienced by transactions, and (3) the rate of fees collected.

The system was simulated by randomly generating both transactions (and their associated values and fees) and blocks according to their respective rates. These transactions and block were then matched with each other (for more detailed simulator explanation refer to modeling approach and assumptions) and pushed out of the system to have their results recorded. The final results were then plotted to allow for easier analysis and visual clarity (refer to appendices).

After analysis of the plots, we found that both increasing the block mining rate and increasing the number of transactions per block would alleviate transaction delays and queues. Additionally, large transaction delays and queues occured where the parameters were set such that $\mu K < \lambda$. Interestingly, parameters set such that $\mu K > \lambda$ by only a small amount can also cause accumulations in the queue and large delays because fluctuations from the random transaction arrivals can generate cases where there were more arrivals than the system can handle.

Therefore, it is our recommendation to set a high value for both block mining rate and number of transactions per block in order to minimize transaction delays and queues. By setting our μ and K such that μK is significantly greater than λ this also allows the system to handle a

sudden influx of transaction requests without creating further delays. However as mentioned in the Conclusions section there are some caveats to simply maximizing μ and K. We want to set values for μ and K that are large enough to handle the overall transaction volume while still having some flexibility to handle a large than normal volume of transactions, but we also want to set μ and K to be the minimum values for this to be possible.

PROBLEM DESCRIPTION

The BRC system relies on a blockchain to maintain consistency, security, and ease of access. Every student in the system is able to make transactions with others granted they have enough funds. The blockchain will keep track of all transactions as well as encrypt data to ensure data safety. It relies on database replication The main concern for the system is that a long transaction queue would generate from a large amount of users.

We are tackling the problem of managing delay times of transactions by seeing how different parameters change throughput. Our goal is to ask for a fair, generalized fee to compensate miners for generating blocks for the blockchain.

MODELING APPROACH AND ASSUMPTIONS

The model that we have constructed is operating under several assumptions. Processing of any existing transactions by a particular block with free transaction slots is assumed to be instantaneous because the effects of processing ability is not a parameter that we are investigating. We are also assuming a block is pushed only when it has been filled with K transactions.

In our simulation set-up, the arrival times of the blocks and transactions were independently generated using the *numpy.random* package and stored in a *block_times* list and a *transaction_times* list. Each list was generated as a poisson arrival process, from time 0 to a specific end time, where each individual arrival event is an exponential process with rate μ and rate λ for blocks and transactions respectively. The value of each transaction was generated as random integers between 5 and 25 in a separate list using the *numpy.random* package, with the fees list being generated as 1% of transaction value or 2% of transaction value using a try-except

method and *numpy.random* functions. The corresponding indices for the *transaction_times* list, *transaction_values* list, and *fee_values* list denoted the complete information for a particular transaction.

Transactions were assigned to blocks using the following formulation: all the data is contained inside of a master list, with each index of the master list containing the data for a particular block, the block time and associated transaction information. The block data is a list whose first index is the arrival time of the block and the following K indices are lists containing the transaction data, namely the transaction arrival time, transaction value, and fee value. (Fig.1)

$$\begin{bmatrix} [block_{time}, [transaction_{information}], [transaction_{information}], \dots], \\ [block_{time}, [transaction_{information}], [transaction_{information}], \dots], \\ [block_{time}, [transaction_{information}], [transaction_{information}], \dots], \\ \dots \end{bmatrix}$$

Figure 1. Master list formulation with sub-lists of block and transaction information

In order to transform the four lists of simulation data (block time, transaction time, transaction value, and fee value) into this master list a series of nested while loops was utilized. The outermost loop would run until the list of block times was empty, thus signifying that all simulated blocks have been pushed. In each cycle of the loop, the first indice of the block time list was popped and added to the block list (see Fig. 1 for list construction). A list was then initialized with all transaction times remaining in the *transaction_times* list before the block arrival time, this is the transaction queue. Lists of the corresponding transaction values and fee values were also initialized. A second while-loop ran which identified the highest fee values in the queue and popped the corresponding transaction's information into the block list as a *transaction_information* list. This loop ran until the queue was empty or until K transactions were placed into the block. In order to deal with the edge case that there were less the K transactions in the transaction queue we introduced a for-loop that would run when the length of the queue list was less than K. Due to our assumption that processing transactions was instant if a block with free transactions slots existed, the for-loop would pop the information of the next transactions to arrive to the block list until the block was filled with K total transactions, at which time it would be pushed.

In this way the nested while-loops ran through all block arrivals and all transaction arrivals, popping the next arrival into a block information list that would then be appended into the master list once K transactions have been filled. This proceeded until the *block_times* list, *transaction_times* list, *transaction_values* list, and *fee_values* list were all empty.

Because generating all of the values, organizing them into a master list, extracting data from that master list and then creating plots with that data took far too much processing power, we instead put all the code into a function of μ and K that would modify and save important data while also building the master list, and then output that data in a form that could immediately be plotted. This ran fast enough for us to be able to use it in a loop that plotted the data over different μ values.

MODEL VERIFICATION

Before creating the model to emulate blockchain flow, we first follow project parameters. We have three design parameters: λ, μ, K. λ describes the transaction arrival rate and μ is the block mining rate. K is the number of transactions per block, chosen from the highest transaction fees among all transaction candidates. A block also needs to wait for K transactions to fill the current block before being processed. There is also a time parameter which is the end point for the simulation. While we largely ignore the intricacies of the blockchain itself, the face validity is still high. The parameters and model structure create a model that accurately mimics a blockchain processing flow.

The model itself is as follows (fig 2.):

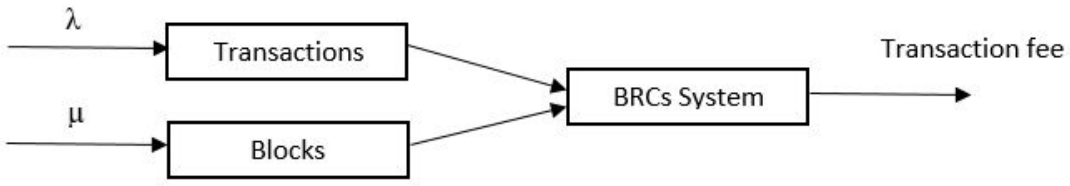


Figure 2. Relationship between Transactions entering and Blocks added to BRC System

We then validate data and structure assumptions. This was done through testing how well our metrics compared to expected results. Our first metric was to calculate the throughput of number of transactions and BRCs. We checked to see if the total number of transactions was close to λ times the total time of the simulation and if number of BRCs was relative to μ, K, and the total time of the simulation. The second metric was delay of transactions being added to the blocks. This calculation was harder to verify, so instead we saw that trends in changes with λ, μ, and K were consistent. For example, increasing K with other parameters constant would increase delay experienced. For rate of fees we collected both the average rate over time and total rate, the latter which is our ultimate deliverable. Since the transaction fee is random, the rate calculation is reflecting the effect of queue times on rate of fees.

MODEL ANALYSIS

Because the two factors under our control are μ and K, we examined how the behavior of the model under different values of these variables. Because K is discrete, only capable of being a positive integer, we created separate plots for each value of K from 1 to 16. On the other hand, μ is continuous, so in order to accurately show the system's behavior with different values of μ, we used 3D plots to show both how the system behaves over time as well as with different values of μ.

Throughput of transactions and BRCs:

The number of transactions and transfer amount of BRCs is directly correlated. BRCs is equal to the number of transactions times a random integer amount between 5 and 25. Below are the plots for throughput of transactions and BRCs

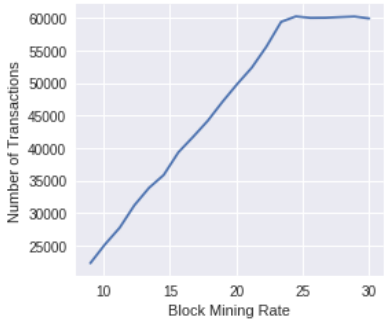


Figure 3. Number of Transactions when K=5

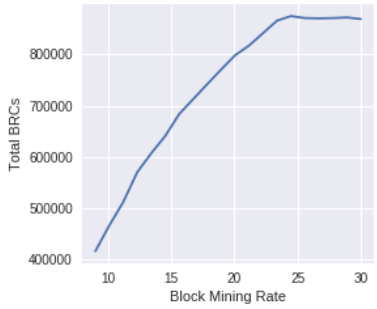


Figure 4. BRCs when K=5

When the K value and μ value of a system are high enough, the number of transactions the system is capable of processing before termination will probably exceed the number of transactions generated. If this is the case, the throughput of both transactions and BRCs is determined by the random numbers generated at the start of the simulation. The resulting plots are random but tightly concentrated. For large values of K, like 16, this describes the whole graph. For smaller K values, such as 9 or 10, this is only occurs above a certain μ value. Below this specific μ value, the system is no longer able to process all of the generated transactions. Instead, the throughput is dependent on the rate at which the blocks are generated. The faster blocks can be generated, the closer the system comes to processing every transactions. This is why the plots show a positive correlation between throughput and μ. Likewise, having blocks that can process more transactions also increases throughput. That's why for any given μ value, a system with a higher K value will have a greater throughput (up until the system is capable of processing all transactions before termination, in which case throughput becomes dependant on the number of transactions generated). The point where a system becomes capable of processing all transactions before termination (if it is visible in the plot) is where $\mu \approx 120/K = \lambda/K$. All of the plots in the appendices support this analysis.

Delay experienced by transactions:

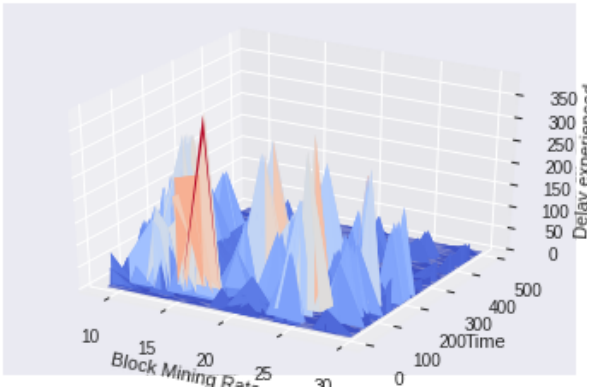


Figure 5. Delay Experienced by transactions when K=1

In Figure 5 we see a 3-D plot of block mining rate relative to delay experienced over the time of the simulation when K=1. The BRC system experiences low delay times when the block mining rate is high. The K=1 input means that every block is filled with one transaction and instantly added to the blockchain. Thus, a slow mining rate will create a bottleneck in transactions. A one-to-one relationship between block mining rate and incoming transaction rate is established where increasing block mining rate and decreasing transaction rate will lower the delay experienced.

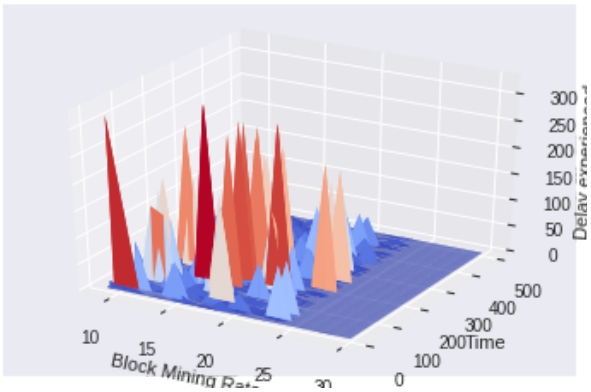


Figure 6. Delay Experienced by transactions when K=5

With K=5 in figure 6, we see that the low rate of block mining contributes to very high delay times because the transactions are coming in at a rate faster than blocks are mined. There are spikes in delay times because of the slow mining rate. But at $\mu = 25$, the transactions experience no delays because blocks are fast to enough to full transaction demand. In equation form, $K*\mu >= \lambda$.

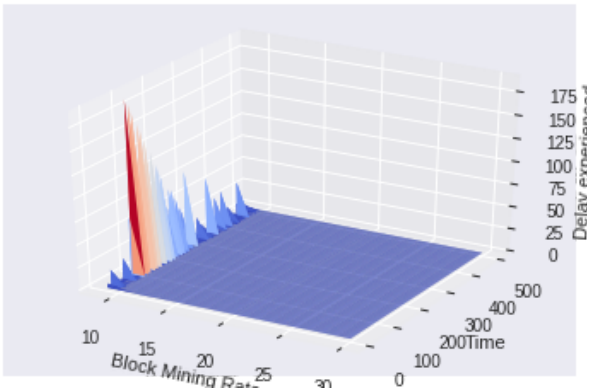


Figure 7. Delay Experienced by transactions when K=13

At K=13 in figure 7, we see that the 0 delay threshold has decreased to the lowest block mining rate of $\mu = 10$. At that blocking mining rate, there are still significant delays. Since the block needs to wait for K transactions before being added to the blockchain, initial delays will be long.

From this, we can see that transaction delay depends in large part on the capacity K of each block. When capacity is low (Fig. 7), the delay experienced is always high due to each block simply not having the capacity to fulfill the transaction demand.

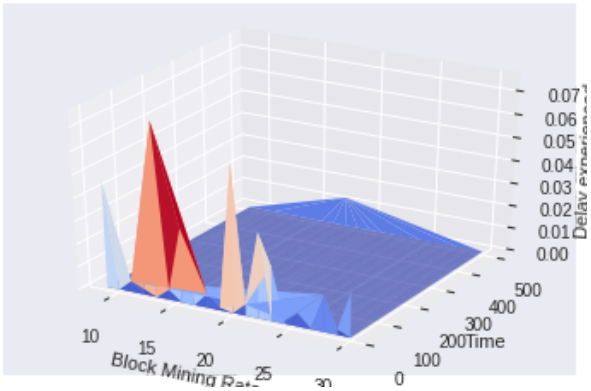


Figure 8. Delay Experienced by transactions when K=16

Finally at K=16 in figure 8, delay only depends on time in simulation. Because the scale of the axis is smaller, we can see very short delays at early times. However, looking closely at the other plots, we can see that the tendency for delays to occur early on in the simulation before decreasing is present in all of them. This is because when a simulation begins, the number of transactions generated is equal to the number of blocks generated. This makes the possibility of the number of transactions surpassing the number of blocks and causing delays more probable,

even if the blocks are actually being generated faster. However, as time passes, the gap between the number of transactions and the number of blocks increases, so the expected delay time decreases. All of the plots in the appendices support this analysis.

Rate of Fees Collected:

The rate of fees collected is also affected by block mining rate μ and block capacity K.

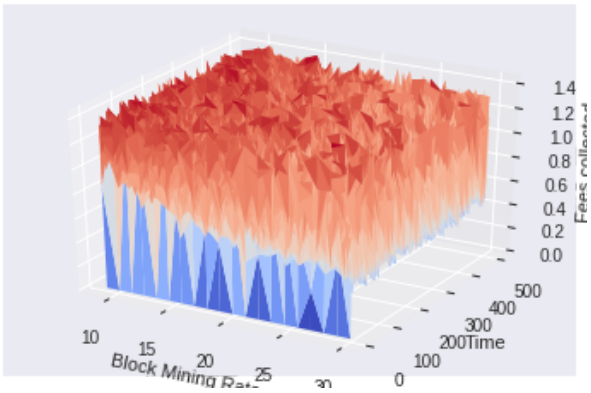


Figure 9. Fees Collected when K=1

In the above figure, K=1 and the fees collected appears to decrease slightly as μ increases. Because miners favor higher fees, when there are more transactions than room within blocks, the miners can be more selective and so more transactions with high fees will be picked. This is especially noticeable in the figure below. Even though we know from the throughput graphs that higher μ values lead to more fees collected overall, these plots reveal that when the system processes more transactions, transactions with low fees are more likely to be processed, thus it appears that the amount of BRCs collected in fees is less, even though it's actually more.

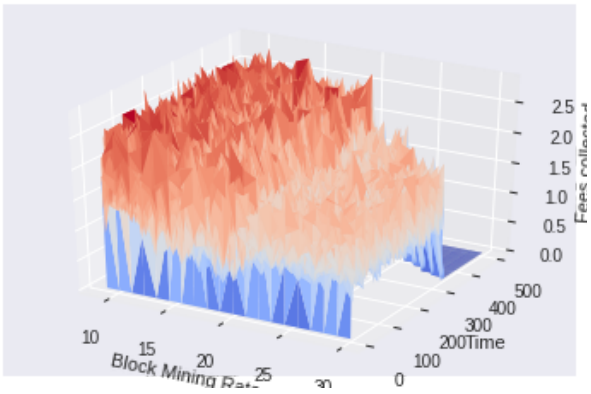


Figure 10. Fees Collected when K=6

When K = 6, we start to see declines in fees collected around the μ = 20 mark. This signifies a drop off in queue length for transactions as there is more room in blocks to accommodate incoming transactions. Past μ = 20, near time = 400 hours, there is a complete drop off in fees collected as there is no longer demand for space in blocks. This is an artifact of our assumption that each block must be filled with K transactions before being pushed as well as our simulation generating a all blocks and transactions for a time frame prior to assigning transactions to blocks.

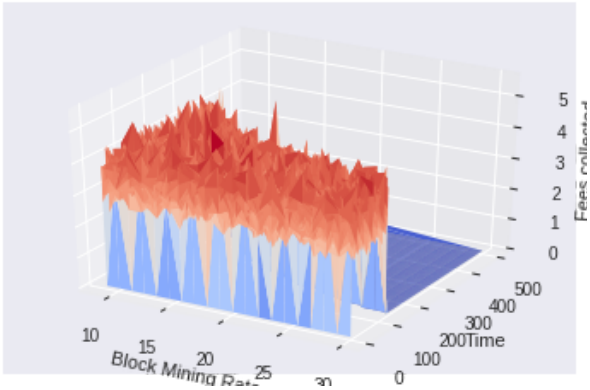


Figure 11. Fees Collected when K=16

And with K=16, the drop off occurs at a faster timestamp as μ increases. As the rate of blocks mined, paired with a large K, matches the rate of transaction arrivals, the queue length goes to 0 and the fees collected also goes to 0.

From this data, we see that as transactions per block, K, increases, there becomes a point where there are simply not enough transactions to fill in the remaining blocks. E.g. for a K value of 16, when μ=16, the last used block is generated at approximately time t=250 and any further blocks are generated but simply not used due to all transactions being processed by blocks mined before that t=250 mark. All of the plots in the appendices support this analysis.

CONCLUSIONS

Based on the data obtained from our model it seems desirable to have a higher block mining rate, μ, and allow for a higher number of transactions per block, K, if minimizing transaction delay is our objective. It is interesting to note that if you set your parameters such that $\mu K < \lambda$, a large transaction queue will accumulate and delays will drastically increase. Even setting your parameter such that it is close to $\mu K = \lambda$ can cause accumulations in the queue and large delays because fluctuations from the random transaction arrivals can generate more arrivals than the system can handle. Therefore, we recommend set a high value for the block mining rate, μ, and number of transactions per block, K, because this would lead to the smallest transaction delays and allow the system to handle more transactions if there was a sudden influx of new transaction requests.

In the real world, increasing μ and K comes at a cost. Making μ too high will make it so that blocks are mined too easily and this may lead to devaluing of the cryptocurrency. If we set K to be too high and must wait to completely fill a block with K transactions, a large backlog may occur initially as we wait for enough transactions to come in. If we wanted to minimize μ and K, but still have a functioning system, we need to determine the boundary between functioning and nonfunctioning systems. If a functional system is one in which all transactions are processed, then we know from examining the throughput plots that this boundary is, as we predicted, $\mu K \approx \lambda$, but such that $\mu K > \lambda$ by some amount that allows the system to handle a larger than average transaction volume.

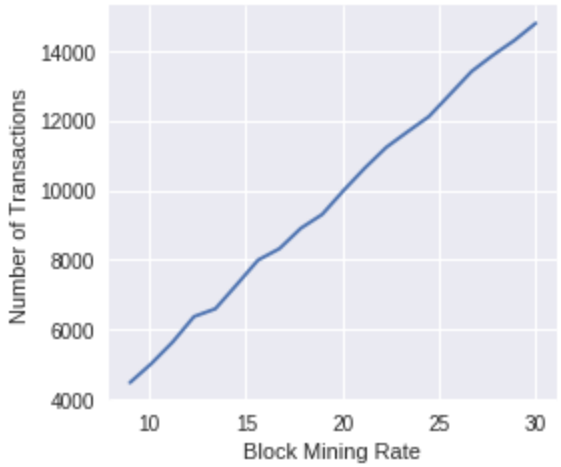
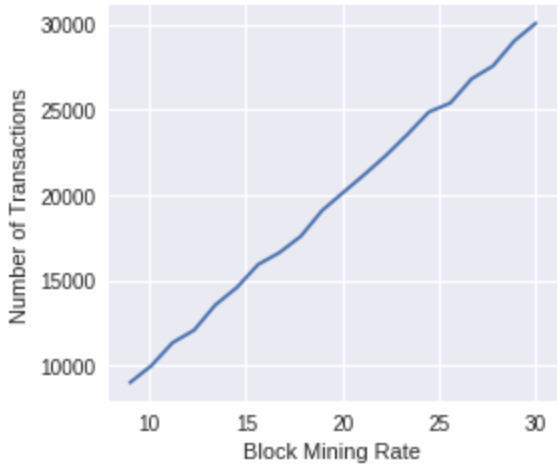
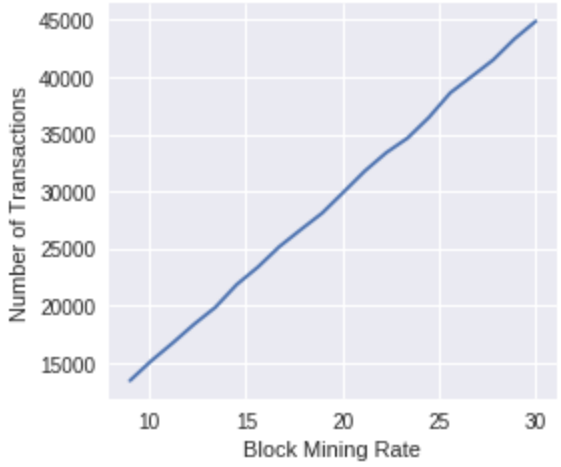
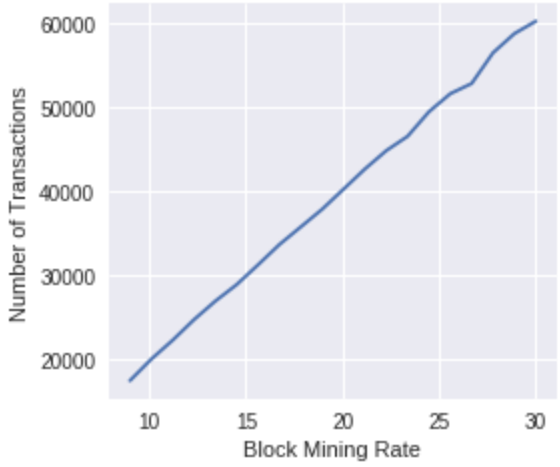
APPENDICES

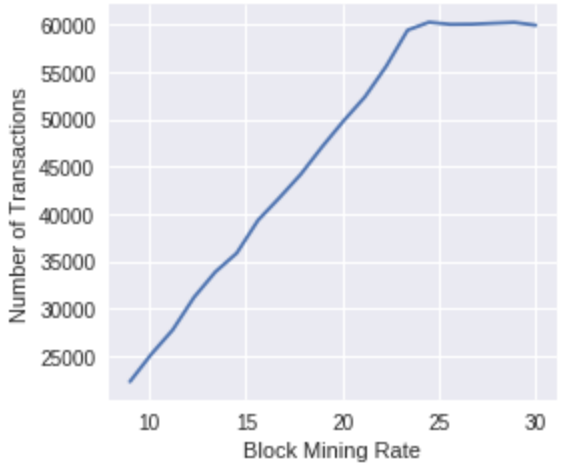
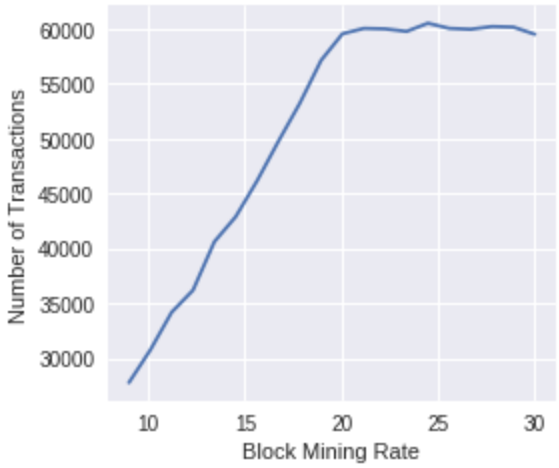
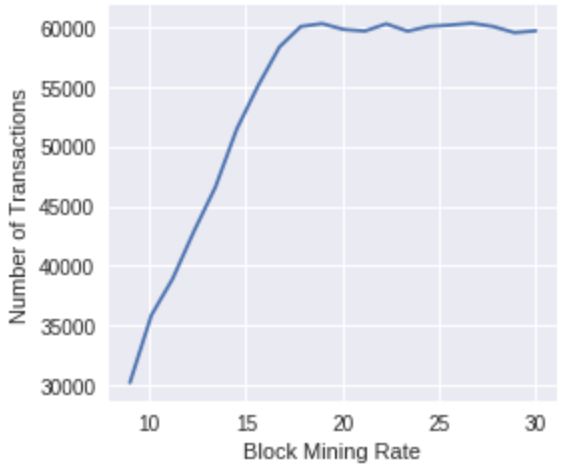
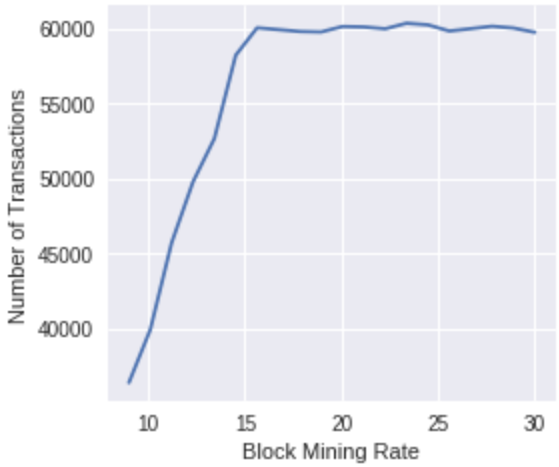
Plots for our various simulator runs are shown in this section.

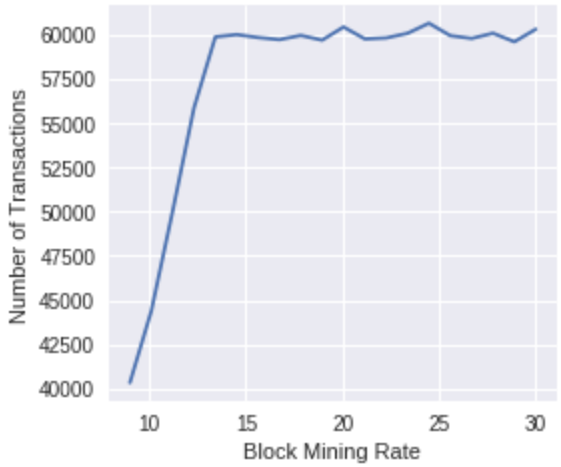
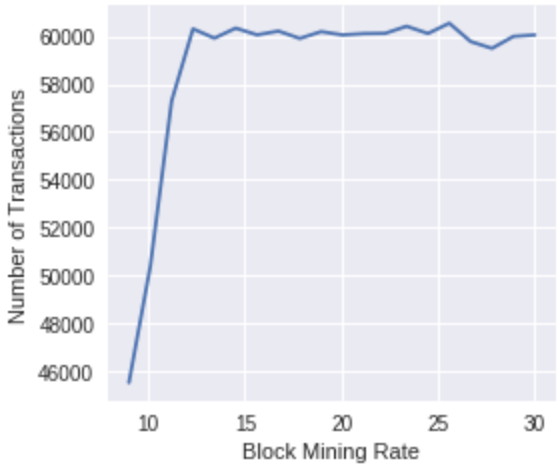
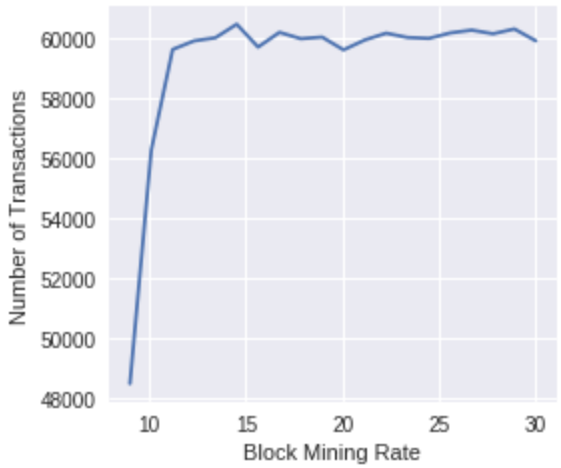
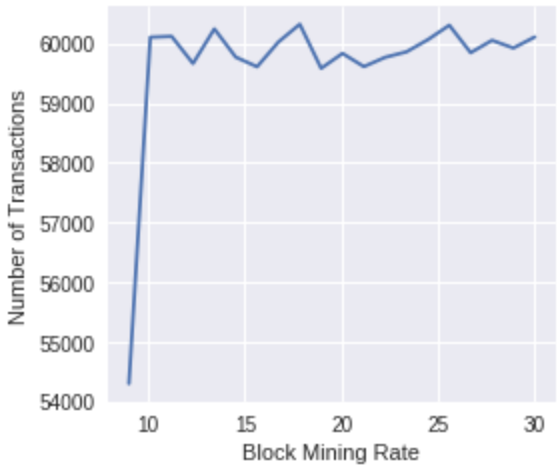
Our code consisted of two function definitions and repeated calls to a function. First, the function Simulate(μ, K) ran the simulation described in the Model Approach And Assumption section. However, trying to use the resulting master list right after creating it took too much runtime, so we collected the data we needed while the master list was being built. To track throughput, we used counters that increased every time a transaction was "processed" by being added to the master list. During that process, information about the transaction, such as the time it was generated and its amount are put into a list that represent that transaction. A block would be represented by a list that contained the time it was generated and all of the transactions in it. Whenever a transaction would be put into a block, we would compare the transaction's generation time with the block's generation time in order to find out how long it was delayed. We then added that delay time to a list that the function would output, and added the transaction's generation time to another list that the function would also output (we actually only output every 50th value to save runtime). Thus, we were able to immediately put these two lists into a plot showing experienced delay behaved over time after running the simulation without modifying or reorganizing any data. We did the same thing to plot fees collected over time as well (only with every 10th value).

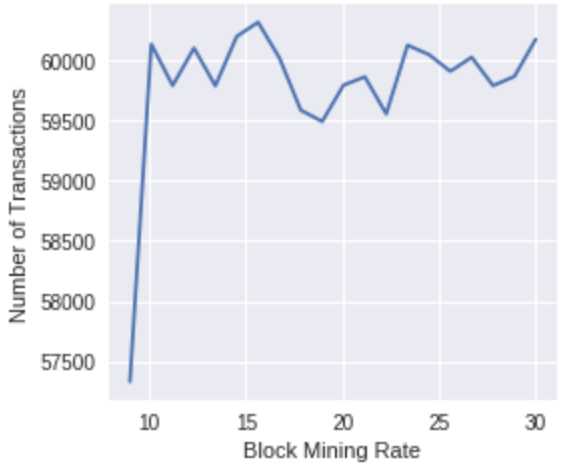
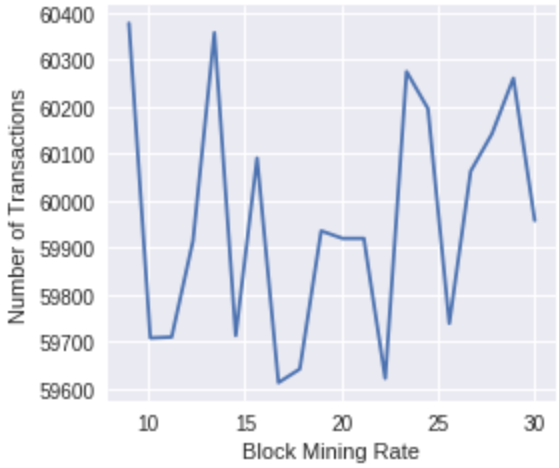
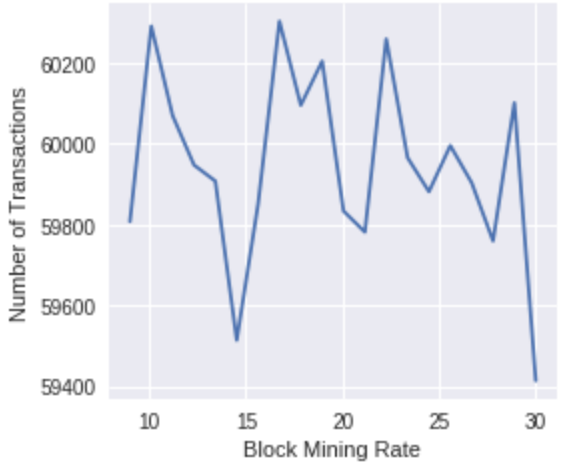
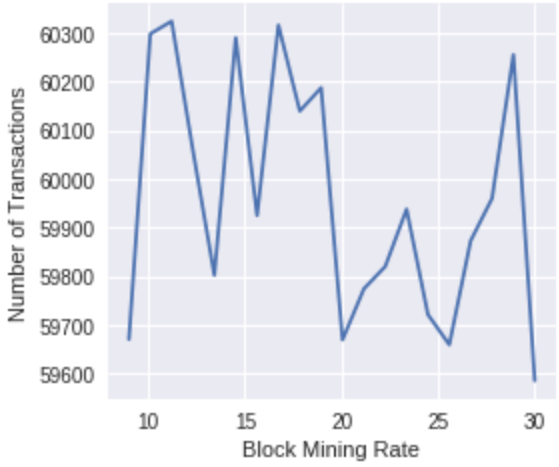
Our second function, Output(K, a), created a list of μ values, then ran a for loop that ran Simulate(μ, K) for every μ in the list. The variable a, which could be 1, 2, or 3, would determine which plot to create (as creating all three at once would take too much runtime). Output(K, 1) plotted both throughput graphs against the list of μ values. Output(K, 2) created a 3d plot of delays vs time vs μ. Output(K, 3) also created a 3d plot of fees vs time vs μ. We ran Output(K, a) 48 times to generate all of the plots we used. Sometimes the runtime approached 5 minutes.

Throughput of System (Number of Transactions):

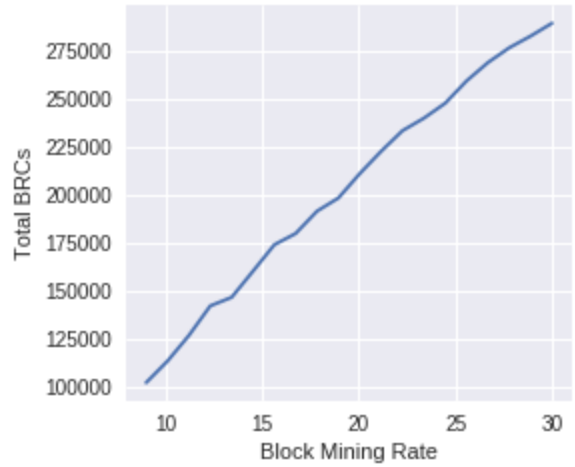
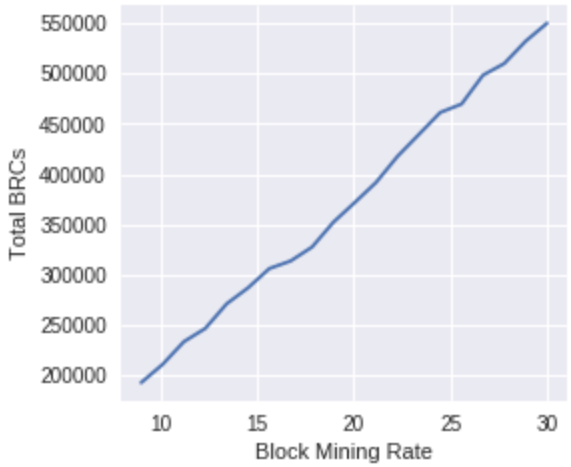
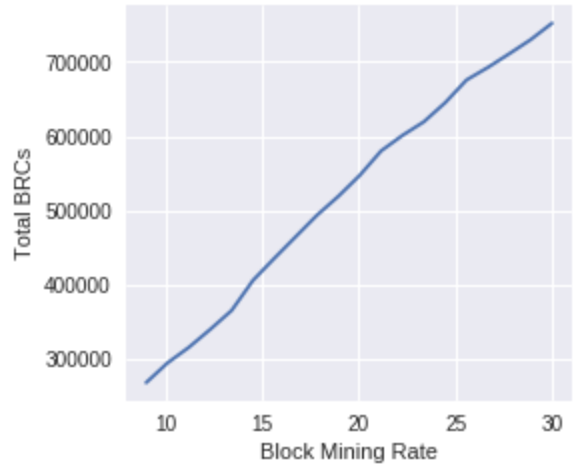
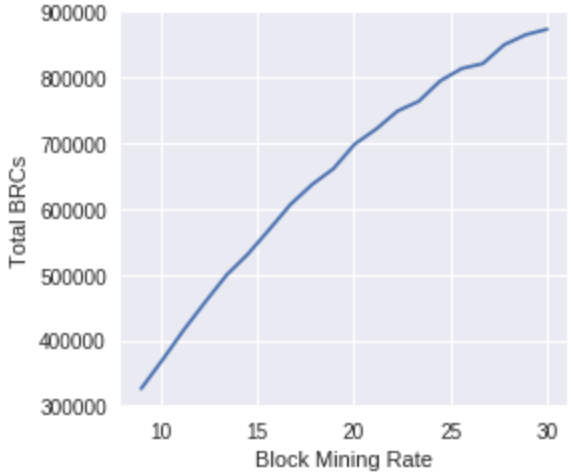
K=1	K=2
	
K=3	K=4
	

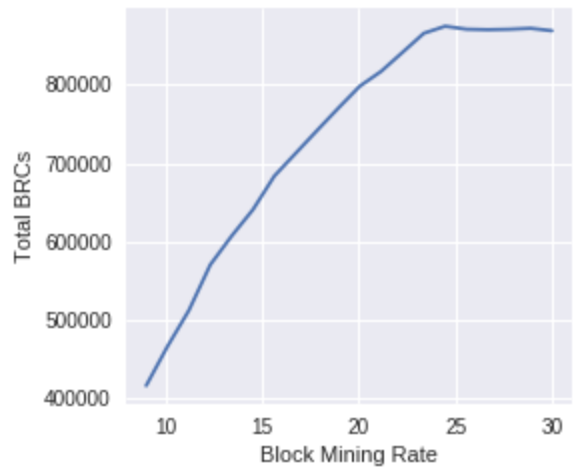
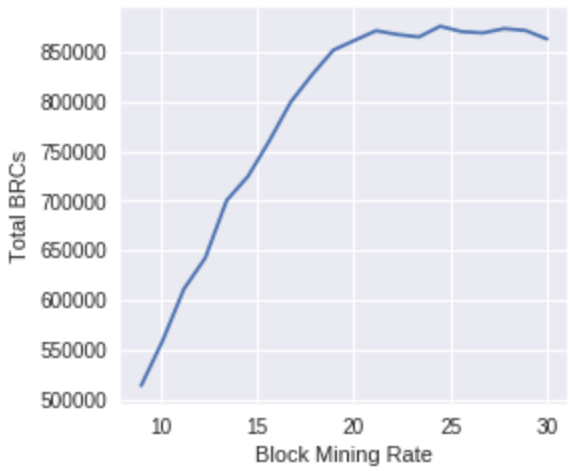
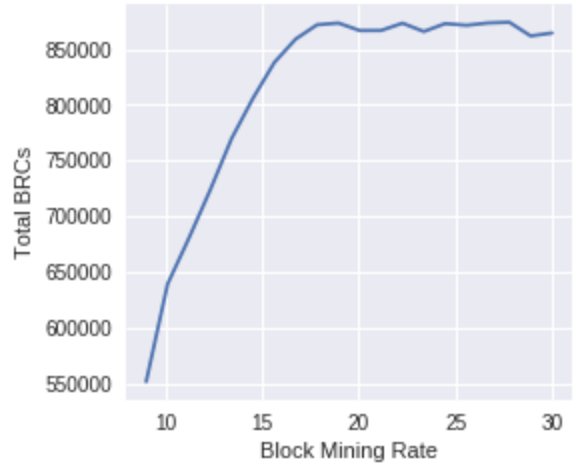
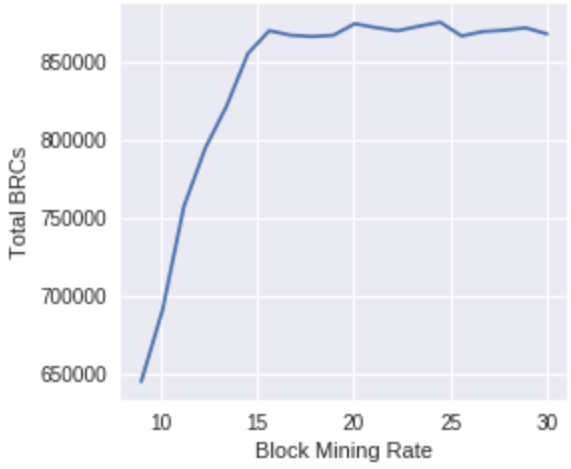
K=5	K=6
 Number of Transactions 60000 55000 50000 45000 40000 35000 30000 25000 10 15 20 25 30 Block Mining Rate	 Number of Transactions 60000 55000 50000 45000 40000 35000 30000 10 15 20 25 30 Block Mining Rate
K=7	K=8
 Number of Transactions 60000 55000 50000 45000 40000 35000 30000 10 15 20 25 30 Block Mining Rate	 Number of Transactions 60000 55000 50000 45000 40000 10 15 20 25 30 Block Mining Rate

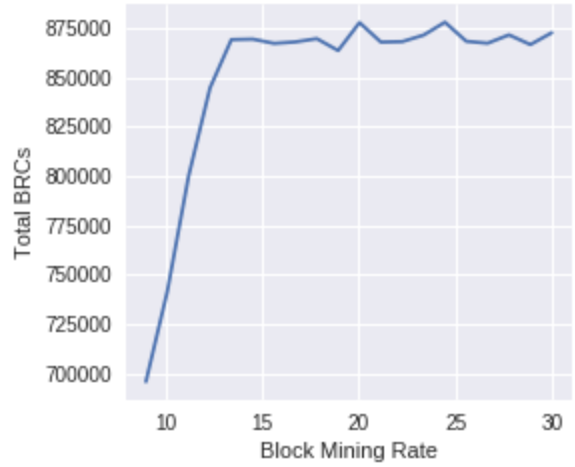
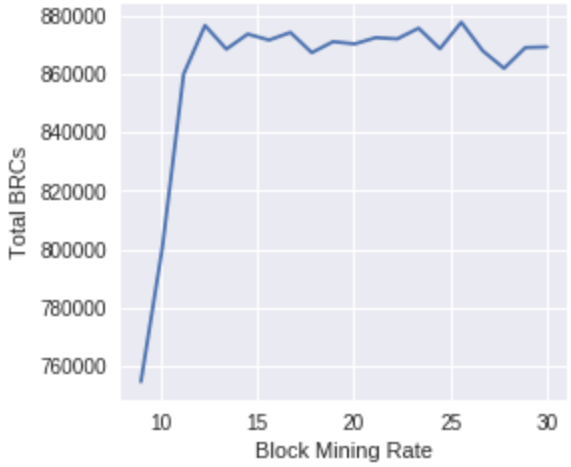
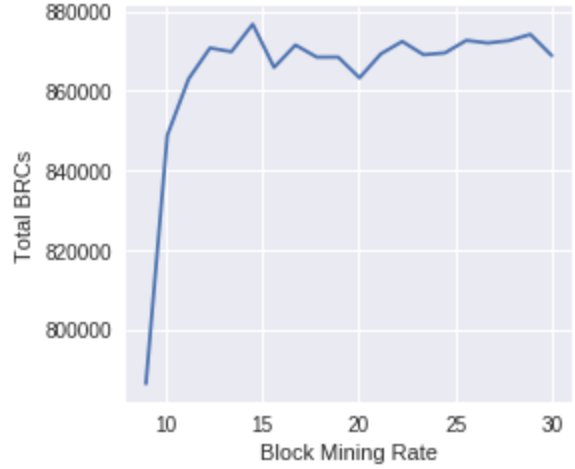
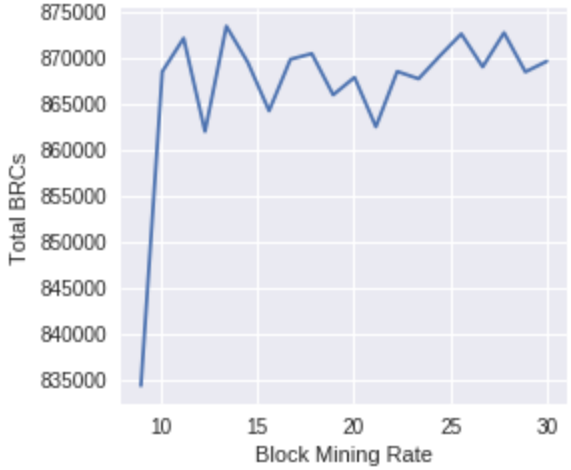
K=9	K=10
	
K=11	K=12
	

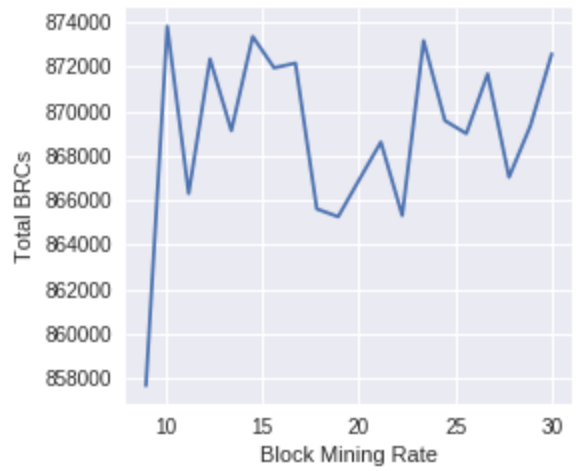
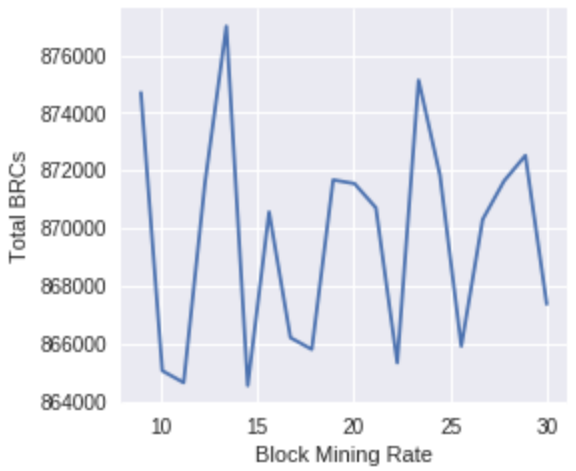
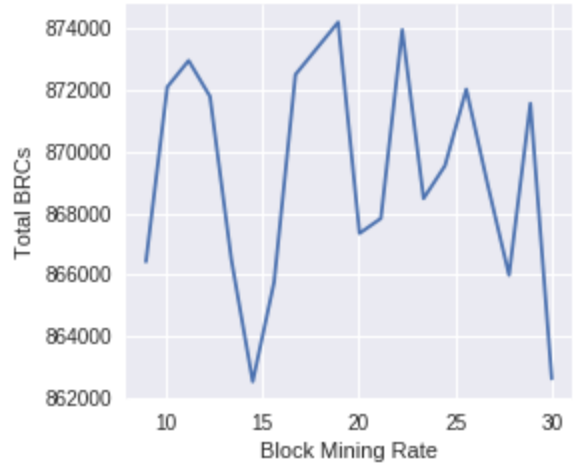
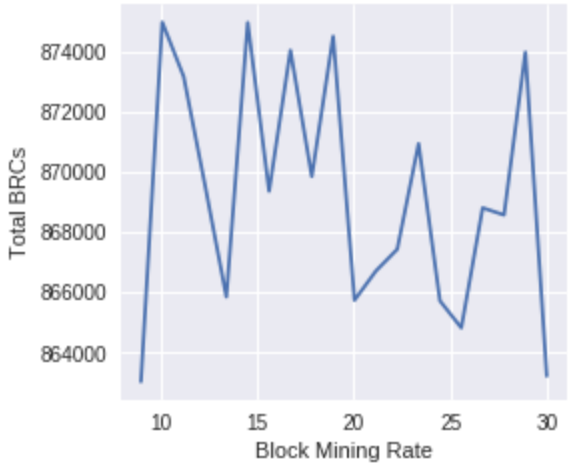
K=13	K=14
	
K=15	K=16
	

Throughput of System (Number of BRCs):

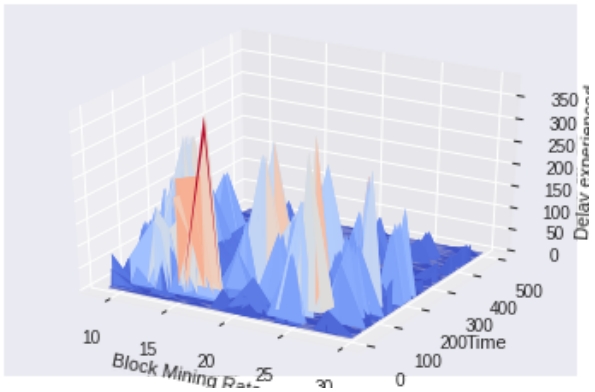
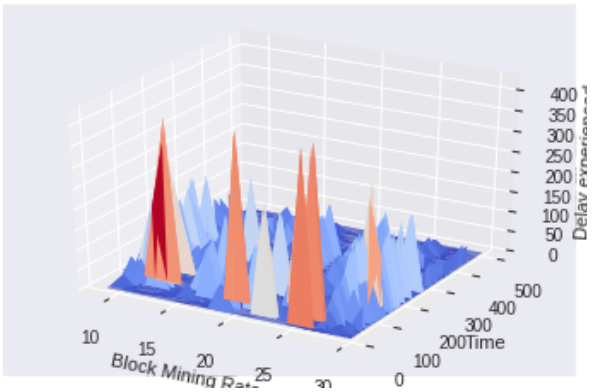
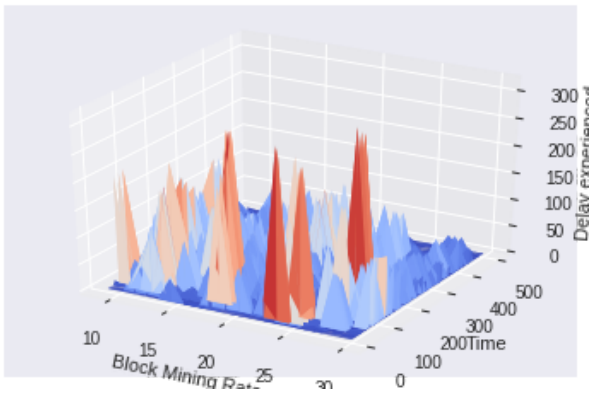
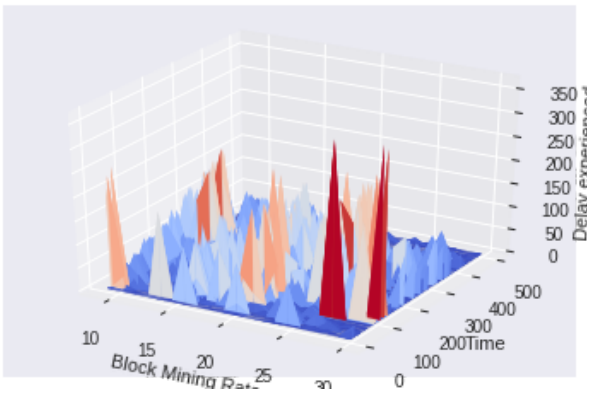
K=1	K=2
	
K=3	K=4
	

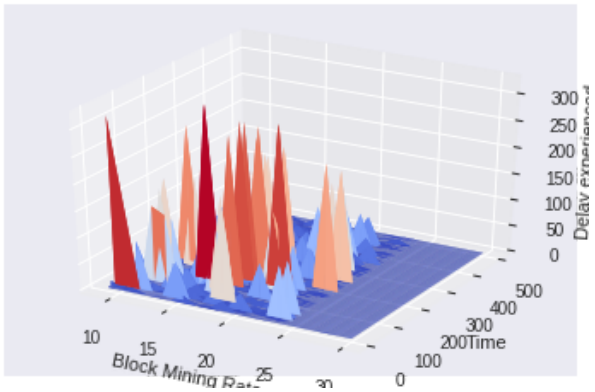
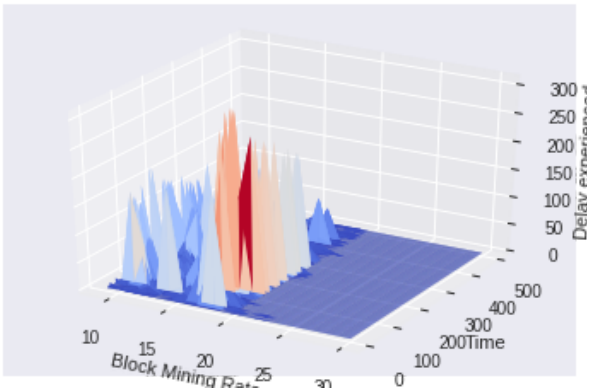
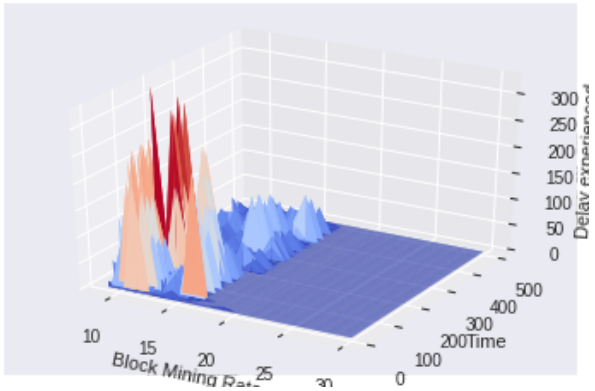
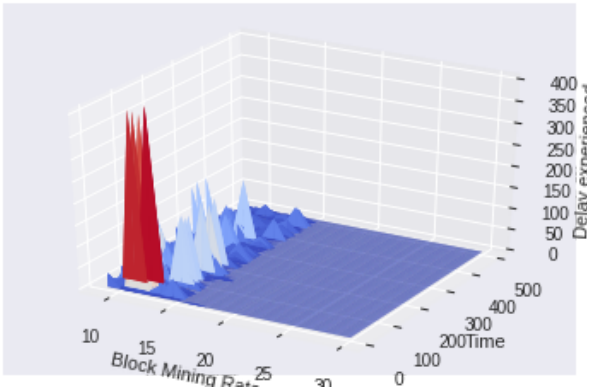
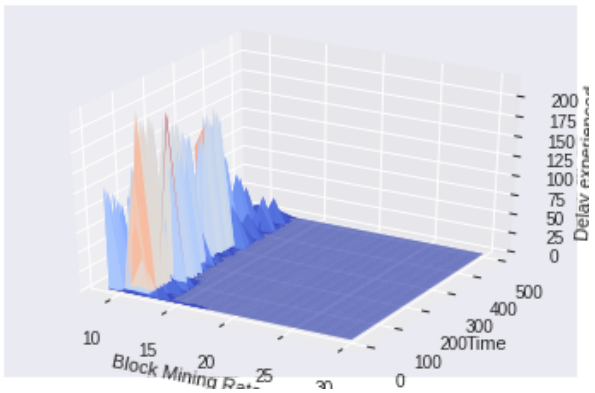
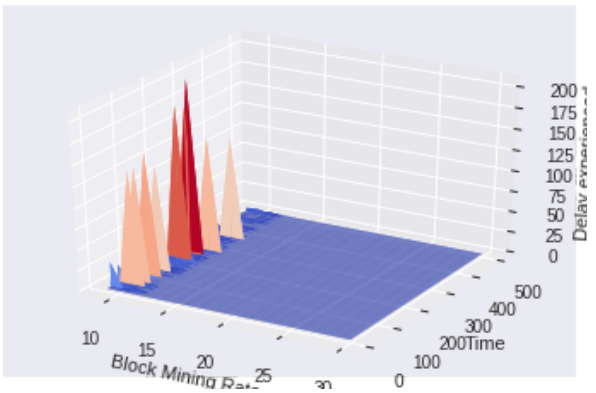
K=5	K=6
 Total BRCs 800000 700000 600000 500000 400000 10 15 20 25 30 Block Mining Rate	 Total BRCs 850000 800000 750000 700000 650000 600000 550000 500000 10 15 20 25 30 Block Mining Rate
K=7	K=8
 Total BRCs 850000 800000 750000 700000 650000 600000 550000 10 15 20 25 30 Block Mining Rate	 Total BRCs 850000 800000 750000 700000 650000 10 15 20 25 30 Block Mining Rate

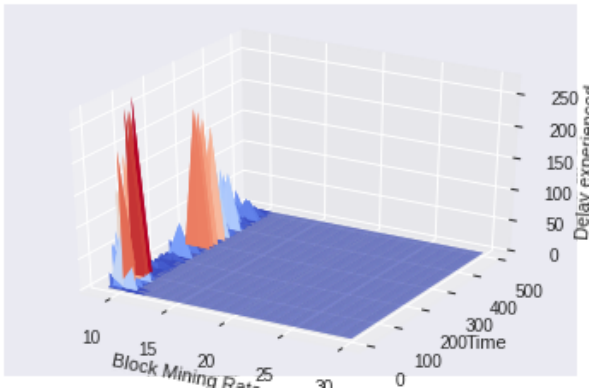
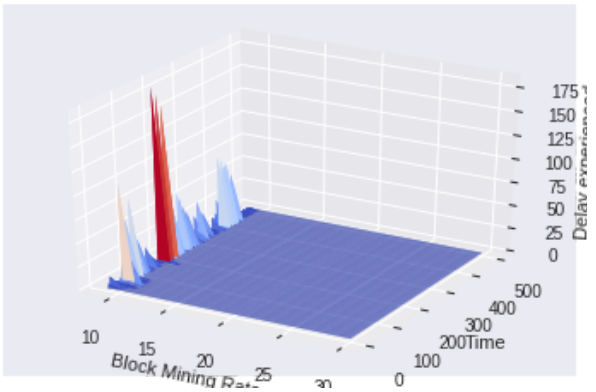
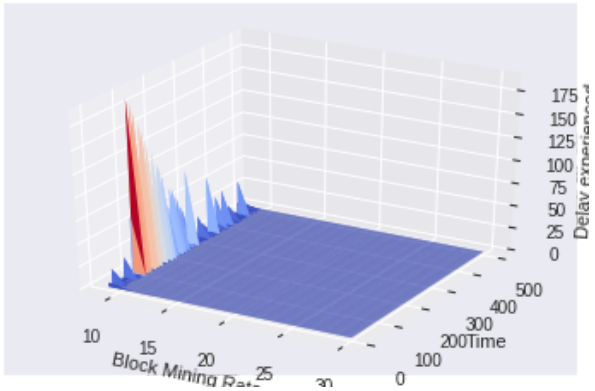
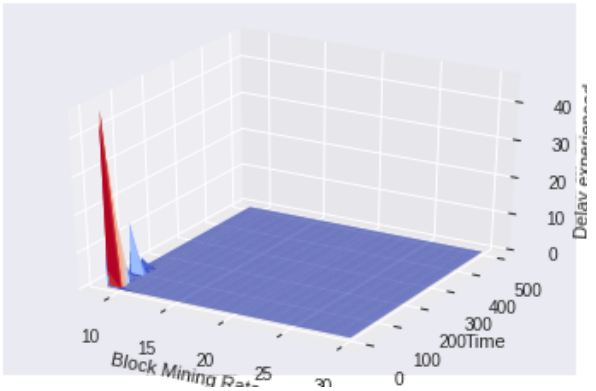
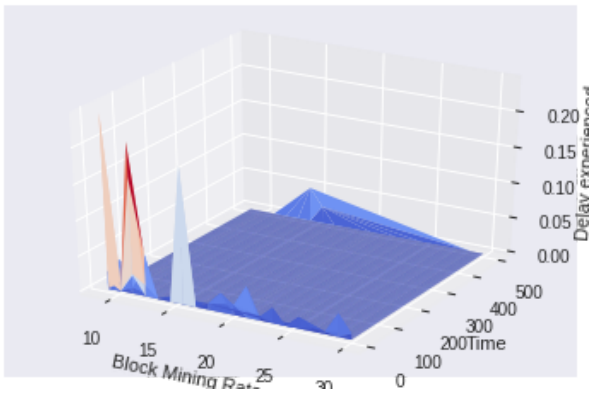
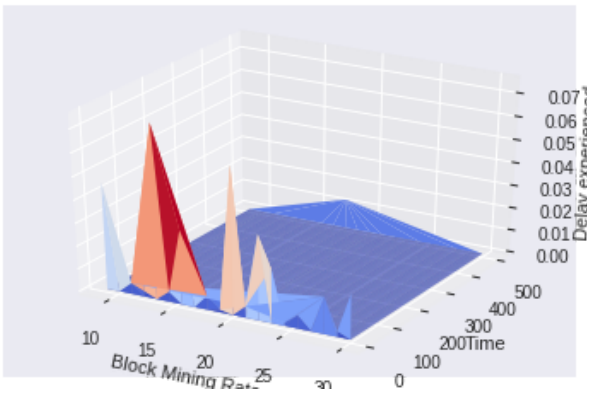
K=9	K=10
	
K=11	K=12
	

K=13	K=14
	
K=15	K=16
	

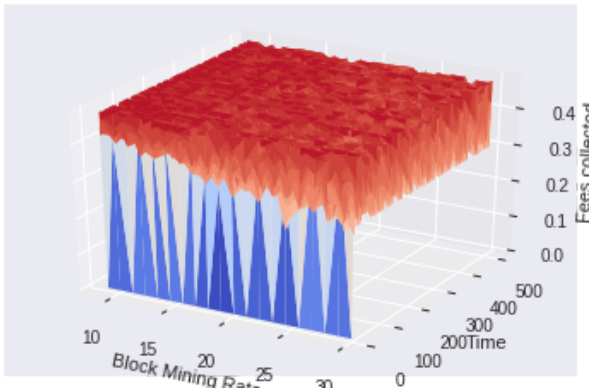
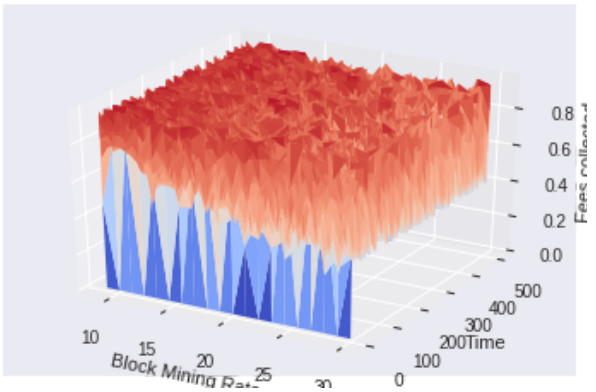
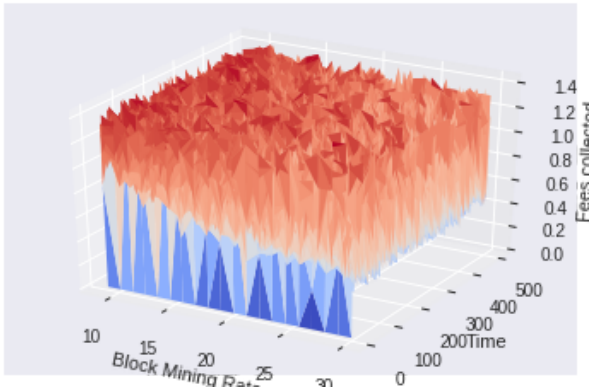
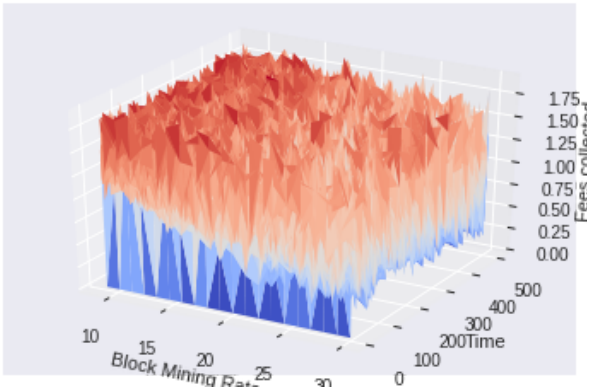
Delay experienced by transactions:

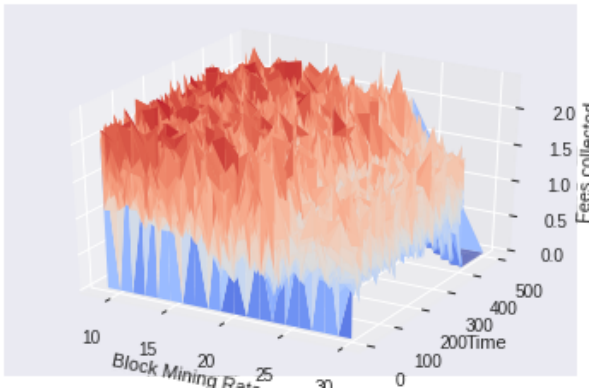
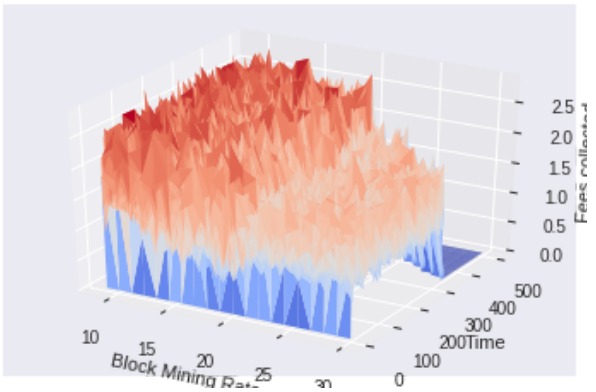
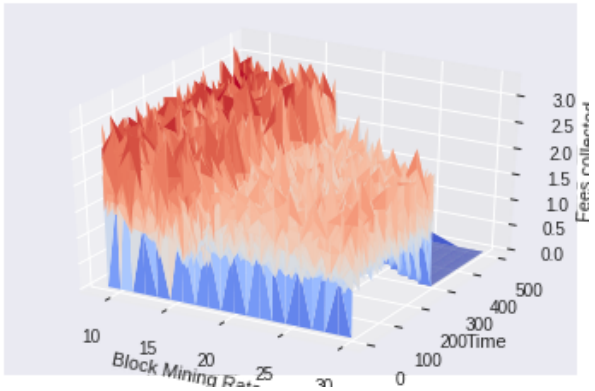
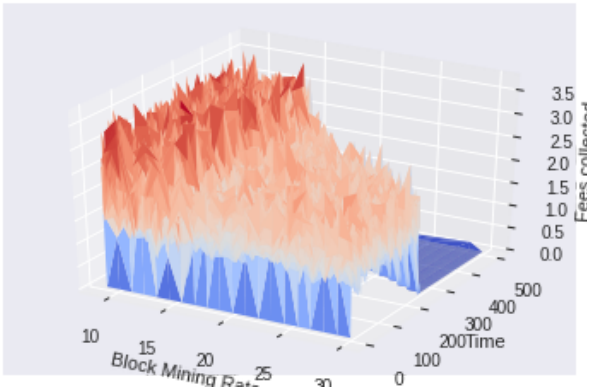
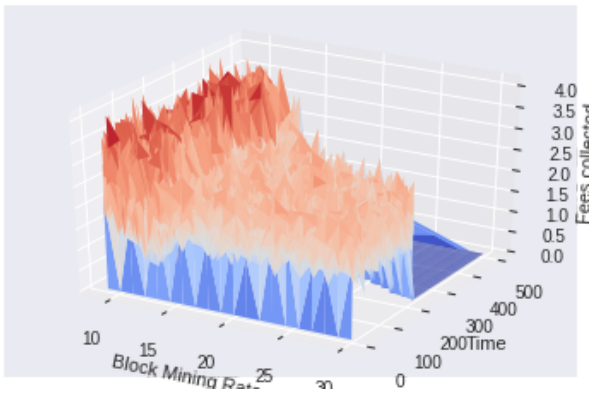
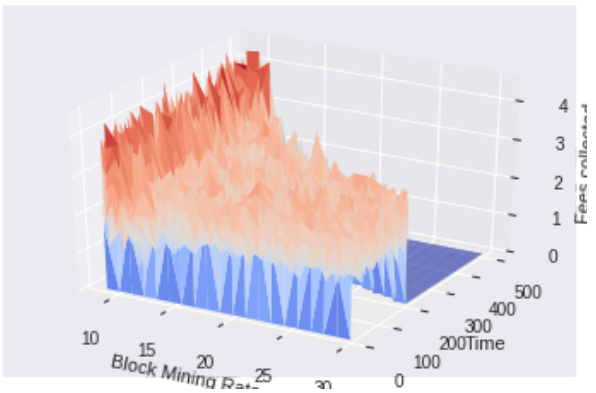
K=1	K=2
	
K=3	K=4
	

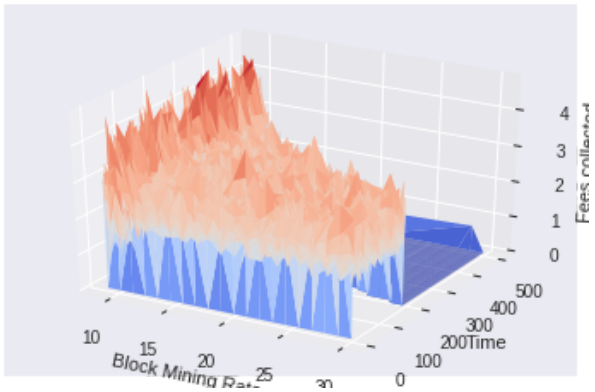
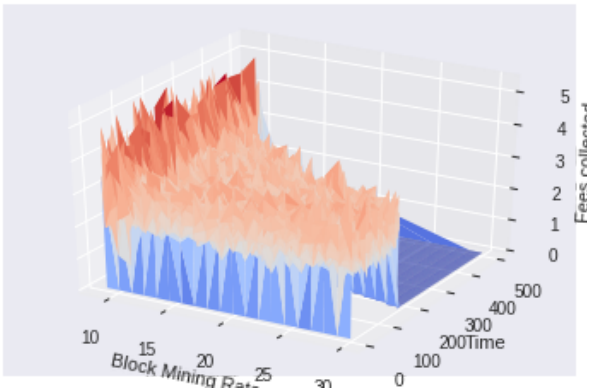
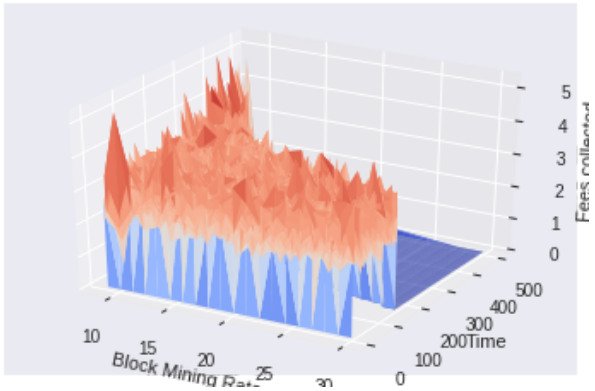
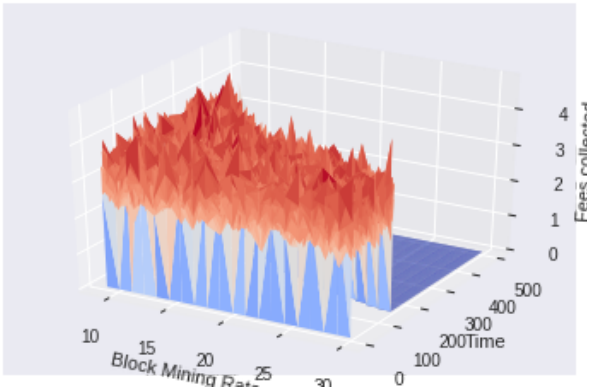
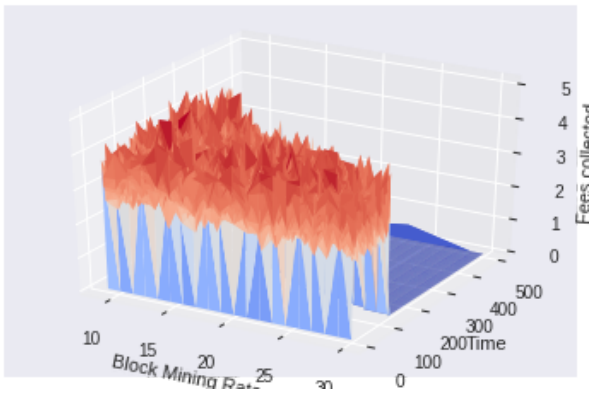
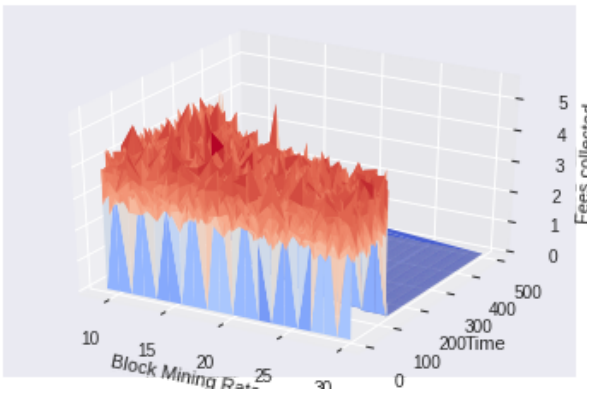
K=5	K=6
	
K=7	K=8
	
K=9	K=10
	

K=11	K=12
	
K=13	K=14
	
K=15	K=16
	

The rate of fees collected:

K=1	K=2
	
K=3	**K=4**
	

K=5	K=6
	
K=7	K=8
	
K=9	K=10
	

K=11	K=12
	
K=13	K=14
	
K=15	K=16
	

REFERENCES

https://www.economist.com/briefing/2015/10/31/the-great-chain-of-being-sure-about-things