Algorithmic Game Theory and Blockchain Consensus Attacks

MATH 486 - Expository Paper

Andrew Millward

May 13, 2021

1 Introduction

At the crossroads between traditional game theory and computer science, a field of studies known as algorithmic game theory emerges. This field is primarily devoted to the interactions present in computer-based decision making with a wide array of applications from protecting distributed Blockchain networks from collusion attacks to design of artificial intelligence systems which often behave as rational agents while making decisions and calculating optimal responses to inputs [1, 2].

In this particular paper, we will specifically look into the idea of adversarial attacks on cryptocurrency networks and how game theory applies in the concept of these real-world issues. In blockchain networks, verification of transactions typically relies on some form of a majority consensus algorithm. If there is sufficiently large computational power across the entire network, it will become impossible for any single individual to overpower the larger network. However, this is not always the case. For example, back in 2018, one actor managed to gain control the majority of Bitcoin gold's available hashing power. Consequently, over the course of several days as the attack progressed, the attackers managed to fraudulently acquire roughly $18 million of Bitcoin gold [3]. Such actions can be very detrimental to not just the credibility of a particular cryptocurrency but the network as a whole, leaving millions of users and their respective exchanges vulnerable to damages.

Since this problem is mostly deterministic by the actions of the entire network and the actions of some arbitrary malicious actor, we can formalize this concept as a form of a Blotto game where, instead of distinct battlefields, players, defined as the entire blockchain mining network as player 1 and the adversarial attacker as player 2, compete to allocate their limited resources in a manner which best suits them. However, in this variant of the standard Blotto game, player 1 has a substantially larger pool of resources to allocate. Player 1 wins if player 2 is unable to win any battle, and player 2 wins if he is able to win in one specific battle (i.e. gains a majority in one currency).

2 Consensus Attacks

It is well known that Blockchain networks rely on some form of consensus algorithm which is determined largely by the computation power of its distributed system. However, should some adversarial party be capable of taking over majority and perform a "51% attack" on any individual cryptocurrency, many transactions could be fraudulently altered in order to gain some economic incentive. As mentioned earlier, one such attack occurred on Bitcoin gold back in 2018.

Across all cryptocurrencies, there exists some cumulative hashing power available from all sources. However, the distribution of this hashing power is often done in a self-serving manner, based on minimizing difficulty and maximizing potential return. This issue leads to an interesting dynamic in a purely competitive environment. As more miners attempt to mine a specific coin, the difficulty of mining that coin goes up, lowering the profitability of that currency. This feature, in turn, pushes miners to other coins that may be momentarily more profitable to protect their own interests. However, if a malicious actor were to arbitrarily drive up the difficulty on a coin at a temporarily loss, they would displace some computational resources within that network. If this attack were completed with sufficient computational power, the malicious actor could hypothetically achieve a 51% majority, compromising the entire network.

In order to prevent this from happening, miners must sometimes act against their own interests to prevent entire currencies from being compromised. While mining capacity typically lies mostly with the most valuable coins (i.e. Bitcoin and Ethereum), the smaller coins remain the most vulnerable. For example, the total hashing power of Bitcoin is roughly 171.8 EH/s whereas smaller coins such as Bitcoin SV (a hard fork of Bitcoin Cash which is itself an offshoot of Bitcoin), has a total network hashing capacity of 745 PH/s. If we consider hardware such as the WhatsMiner M30S++ at 112 TH/s, we need only 3226 units at around $9.2 million to achieve a majority consensus (assuming we get the full 112 TH/s on this currency, where in reality it would likely be lower). This number, while large, is well within the realm of possibility, and for even smaller currencies, could be much easier. In this hypothetical, how exactly should mining networks allocate resources to prevent attacks which invariably affect everyone on the network in the end? To answer this question, we can consider a simulator to compute different allocation schemes and responsiveness levels to such disturbances to determine how a network could respond to some outside attacker.

3 The Simulator

To emulate these characteristics of 51% attacks on cryptocurrency networks, I will construct a simulator to run several iterations of such an attack where one player is the attacker and the other is the network. The code for the simulator can be found at
`https://github.com/andrew-j-millward/Blockchain-Blotto-Game`.

3.1 Players and Strategies

To begin, let's formalize some of the parameters that will be necessary to understand the simulator. Firstly, there will be two players in a repeated sequential game. Player 1 will be the network which will look at resource allocation and then adjust hash rates on overweight cryptocurrencies to increase capacity on underweight ones relative to payoff in order to equalize the network payoff ratios. Player 2, on the other hand, will be formalized as the attacker. As an attacker, Player 2's strategy will be to find the optimal currency within a specific time frame and attempt to allocate his resources into that currency in order to attempt to reach a 51% majority to gain control of one particular currency. This optimization strategy will be accomplished by assigning 100% of Player 2's total hashing capabilities into whichever coin currently has the lowest total hashing power assigned to it by the network, discounting the hashing power of the coin the attacker is currently in. On each turn, both the hashing power vector across each coin's distribution as well as the payoff vectors will be re-calibrated to adjust for all changes to the network's allocations. These two strategies employed by both the network and attacker will be referred to as the network's greedy heuristic and the attacker heuristic.

3.2 Initialization

The initialization of this simulator is relatively straightforward. To start, the simulator parses all input values given by the user. Next, the simulator will take the cumulative hashing power of the network and divide it evenly across each coin (prior to applying the network heuristic for the first time). Then, the attacker's coin will be selected as just any coin without regard for any specific payoff or other factor. Finally, the payoff vector will be randomly generated as an array the size of the number of coins with each element being a decimal between 0.01 and 0.99 with two digits of precision. Following this initialization of the main vectors of interest, the simulator will proceed for 50 iterations with the control flow of network heuristic, attacker heuristic, payoff calculation, and finally market adjusting before repeating.

3.3 Payoff Function

Following each set of player turns, the payoff of that particular round will be calculated. This is accomplished using a simple payoff function. Effectively, there exist two cases for the overall payoff function. In the first case, the attacker has managed to get a majority of the hashing power in one cryptocurrency. In this case, there will be a hard-coded payoff of (-100, 100), and the game will end immediately following this result with the attacker winning and the network ultimately losing. We will consider the network as having lost even though immediate ramifications will only be a sharp drop in value for one particular cryptocurrency and potential loss in credibility for some other cryptocurrencies which could devalue the network assets as a whole. The alternative scenario for the payoff function is that the attacker failed to obtain a majority hashing power at any point. In this case, the payout will simply be computed by taking the dot product of the hashing vector and payout vector for both the network and attacker. This accurately computes the payoff since the dot product will yield the total sum of hashing power times payoff per unit of hashing power for each currency for both players. Consequently, both players will have a non-zero payoff in this case with the attacker's payoff being sub-optimal due to potential for mining a less efficient coin. I will iterate more later on this point in section 5 of this report as this does somewhat relate to how long an attacker can possibly sustain an attack on the entire network before succumbing to other economic factors.

3.4 Market Simulation

The last major factor this simulation will directly consider is the impact of a market environment. For instance, in the real world, a market can cause a divergence in the payoff of particular coins to diverge over time. As a result, variance in the payoff is introduced over the long run. To effectively simulate this principle, this simulator will consider a market function that will give coins the opportunity to increase or decrease in value by a certain percentage between 0.99957 to 1.00057 times the original amount. As you can see, a slight bias is given to the upwards direction which simulates a slight bias upwards in the market pricing over a long period of time. Each element of the payoff vector will be multiplied by a random factor within this range to form the new payout vector that will be used in the next turn.

4 Results

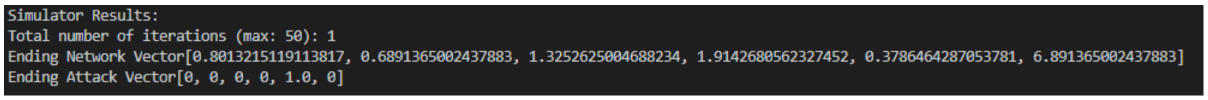
To test the simulator, we will construct a theoretical blockchain network consisting of a total of 6 cryptocurrencies. Suppose the total network hashing power for this network of blockchains, excluding the attacker, is 12 TH/s. Now, suppose the total hashing power of the attacker is 1 TH/s. In the context of this simulator, we can express this state by using the following command:

```
python main.py 6 12 1
```

Now, when we run this command, depending on the random initialization of the network, there exist several possibilities, each of which will get an example screenshot from running the above input parameters on the simulator.

4.1 Immediate Attacker Win

To begin, let us consider the case when the simulator assigns payoffs with large disparities randomly at the beginning. Due to the large disparities, the network will naturally move away in large quantities from the low payoff coins to those with the high payoffs. This leaves a massive potential attack vector for the attacker to take advantage of in his turn. At that point, all the attacker needs to do is exploit the vulnerability in the originally low payoff coin to gain majority control and profit. Such an example can be seen in the following screenshot:

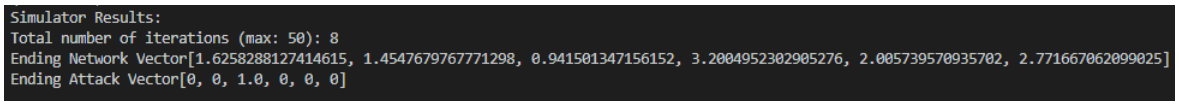


```
Simulator Results:
Total number of iterations (max: 50): 1
Ending Network Vector[0.8013215119113817, 0.6891365002437883, 1.3252625004688234, 1.9142680562327452, 0.3786464287053781, 6.891365002437883]
Ending Attack Vector[0, 0, 0, 0, 1.0, 0]
```

In these results, you can see that the total number of iterations of the simulation was only 1. This implies that the attacker immediately gained control of at least one coin and won the game. Looking at the initial payoff vector that caused those results, we get [0.57, 0.5, 0.74, 0.82, 0.09, 0.95] which corroborates with our theory since the 0.09 value corresponds with the same index that the attacker exploited in the ending attack vector in the screenshot.

4.2 Multiple Rounds

Now, let's look at a more interesting case. The second major case is when the attacker and network get a chance to go back and forth on several occasions before the attacker ultimately wins. One particularly good example of this occurring is present in the following screenshot:

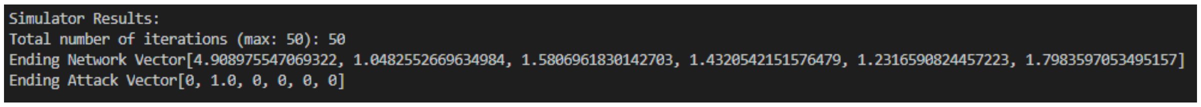


```
Simulator Results:
Total number of iterations (max: 50): 8
Ending Network Vector[1.6258288127414615, 1.4547679767771298, 0.941501347156152, 3.2004952302905276, 2.005739570935702, 2.771667062099025]
Ending Attack Vector[0, 0, 1.0, 0, 0, 0]
```

As you can see above, this simulation made it through 8 entire rounds of the simulator before finally concluding. Such a result could be indicative of multiple factors at play. For starters, in order to get a balance to occur, we know that the differential between the maximum and minimum payoff ratios was large enough to allow the attacker to gain a majority eventually but not enough to grant it outright. The second major factor at play in this demonstration lies in the market function of the simulator. As the market attempts to increase variance in the network, the coin with the minimum payoff ratio managed to be decreased enough relative to the maximum payoff that the attacker eventually won after 8 iterations of the game. As a result, this is a very interesting example of when the game is played for several rounds before terminating.

4.3 Network Win

The final major type of outcome that is present in this game from the simulations is when the network wins. This occurs when the differential between the maximum and minimum payoff ratios is small, preventing the network from overextending itself into one coin at the expense of another coin's security. Below is a screenshot of such a phenomenon occurring:



```
Simulator Results:
Total number of iterations (max: 50): 50
Ending Network Vector[4.908975547069322, 1.0482552669634984, 1.5806961830142703, 1.4320542151576479, 1.2316590824457223, 1.7983597053495157]
Ending Attack Vector[0, 1.0, 0, 0, 0, 0]
```

As you can see, the simulation went through all 50 iterations of allotted iterations before cutting off. Though it is not output directly, the payoff at the last iteration would be upper bounded by $(50 \cdot 7.229137652305607, 50 \cdot 0.6024281376921351) = (361.456, 30.1214)$ as that is the payoff achieved by each after 50 iterations. As part of the future work, it may be advantageous to compute this number exactly as it will vary by a few percentage points due to the random fluctuations in the market price functions. With that said, the ultimate conclusion that this section was looking towards was who 'won' the game under the definitions outlined at the end of section 1.

4.4 Takeaways

The most crucial takeaways from this experimentation is that the outcome of the game largely depends on the payout ratio of the coins in the network. As a result, this implies that smaller coins with lower prices relative to their payoffs and difficulties are the most susceptible to these 51% consensus attacks on the network. That is, the outcome is typically related very closely to the state of the network on initialization due to the fact that the network has an inherent profit focus and will seek to maximize their own gain according to the most efficient distribution of resources. As a direct consequence, this leaves normally less profitable coins as the most vulnerable. Finally, this simulation has also shown the importance of the market fluctuations in price on the abilities of an attacker to exploit the network as seen in section 4.2. Without price fluctuations, the attacker would only be able to win if they had sufficient computational resources immediately when starting, but by allowing coins to adjust over time opens up windows of opportunity due to increased variance in the prices, eventually allowing the attacker to take advantage over a specific coin and gain control.

5 Future Work

There exist several areas of improvement for this work in the future that may be of interest. Firstly, one major area of concern for this simulator would be to develop better algorithms for adjusting the payoff vector on the attacker's turn. Currently, the algorithm runs into some issues where the net payoff is not preserved and will need to use a correcting factor to compensate and return the correct value, but I believe that a well-formulated expression could be made to more precisely accomplish this task. Secondly, the market function currently just trends upwards with a bias. However, in the real world, a realistic market would be able to see large day to day shifts with upwards and downwards trending periods of time. In an improved simulator, the market should also adjust the bias value to more fluidly allow for these periods to occur to provide better insight into real-world systems. Next, the current structure of the program, while allowing for a reasonable approximation of crytocurrencies through payoff vectors, would likely be more accurate were it to use a combination of coin difficulties and reward ratios to better compute the network heuristic. This is because payoffs are an indirect metric to approximate the underlying difficulty and rewards that both vary related to the specific algorithm, coin pricing, and other factors. Finally, I believe this algorithm could be easily tied to real-world data once the above changes are realized. For example, in the past, I have personally worked on a small project to allow for data collection from websites in real time through web-scraping (`https://github.com/andrew-j-millward/Excel_Web_Queries`). As a result, I believe that the implementation of such a scheme to compute the real-time prices of a subset of cryptocurrencies online along with the network hashing power, difficulty, reward rate, and mining pool sizes could drastically improve the usability of this software.

As a whole, the results of this experiment give some insight into the nature of blockchain networks as well as the importance of having a high hashing rate threshold on every currency of considerable value. Should a single coin become unprofitable while relying on a well-developed hashing algorithm, that coin could be at risk of one of these attacks occurring from a sufficiently funded entity with intent to gain a majority as what happened with Bitcoin Gold in 2018 [3]. This project provides a robust framework from which a more accurate model can be constructed with the above suggestions to better consider, detect, and deter these attacks from happening in the future.

References

[1] Dey, S. Securing majority-attack in blockchain using machine learning and algorithmic game theory: A proof of work. In *2018 10th Computer Science and Electronic Engineering (CEEC)* (2018), pp. 7–10.

[2] Elkind, E., and Leyton-Brown, K. Algorithmic game theory and artificial intelligence. *AI Magazine 31*, 4 (Sep. 2010), 9–12.

[3] Roberts, J. J. Bitcoin spinoff hacked in rare '51% attack', May 2018.