Simulation of a Selfish Mining Attack in a P2P Cryptocurrency Network

CS765: Introduction to Blockchains, Cryptocurrencies, and Smart Contracts

Sourab Jha
19305R003

October 17, 2021

The task in this assignment is to prepare a discrete event simulator for a P2P cryptocurrency network, which simulates two variants of the Selfish Mining Attack as discussed in [1]. I have used Python for this simulation. The implementation details follow.

1 Initialization

At the very start, we calculate the fixed delays corresponding to each pair of nodes, which will help us in calculating latencies later.

We also create a network graph that resembles a real-world cryptocurrency network. Existing research ([2] [3] [4]) suggests that cryptocurrency networks (and even the internet, in general) can be best represented by a small-world graph, where the degree distribution of nodes follows the Power Law. In accordance with this, I have used the Albert-Barabasi algorithm [5] to generate a random connected graph containing n nodes. Each of these nodes is a peer of the cryptocurrency network, represented by integers 0 to $n-1$. Each of these nodes is marked slow with a probability z, or fast with a probability $1-z$.

Finally, we also pre-populate the event queue with a set of m transactions from each of the n nodes. These transactions of the form "$TrxID$: A paid B an amount of K coins." For each A, m Bs are randomly chosen out of the remaining nodes, with replacement. The amount K is sampled from an uniform distribution with mean 10. The generation of the transactions can be thought of as a Poisson process. Hence, the inter-arrival times of the transactions are sampled from an exponential distribution with mean i.

These transaction events are added to a queue (implemented as a heap), and subsequently created events are added to the same queue.

At the start of time, we also schedule mining events at all nodes with a mean completion time m, and add these events to the queue.

2 Transaction Broadcast

Execution of each transaction happens as follows: for ease of processing, the node generating the transaction sends the transaction to itself at the same timestamp it is generated. Upon receiving any transaction, the node checks if it has seen the transaction before. If not, it sends it to all of it's neighbour (except the one it received the transaction from). The reception events are queued for each nodes after a time delay corresponding to the network latency between the two nodes, which is given by $\rho_{ij} + \frac{|m|}{c_{ij}} + d_{ij}$, where i and j correspond to the source and destination nodes, ρ_{ij} to the speed of light propagation delay, c_{ij} to the link speed between i and j, and d_{ij} to the queuing delay at node i. d_{ij} is inversely proportional to c_{ij}. This is because the faster the link speed is between the two nodes, the lesser would be the probability of the message getting stuck for longer periods in the queue at the source, since the source would not be busy propagating the earlier queued messages.

3 Block Creation and Broadcast

Two different strategies are followed for block creation and broadcast. One is for honest nodes and the other for the selfish adversarial nodes. Unless specifically mentioned, most features of this algorithm are common to both types of nodes.

Each block contains information about a set of transactions, about the creator of the block, and a pointer to it's parent block. At the start, each node has the genesis block (which, in the likeness of bitcoin, has a non-existent peer as it's creator, which is denoted by -1).

The passing of created blocks is exactly the same as the passing of transaction messages, except the actions taken by the receiving node after receiving the block.

Each node keeps track of the blocks received by it. If the received block forms a new longest than previously seen, the node starts a new mining event over this new block. While executing the mining event, if the block's parent is no longer the longest chain seen by the node, the new block scheduled for mining is discarded. While this discarding should be done whenever a new block is received, in this case it is being discarded at the end of the mine time so that the event queue (which is a heap in this case) does not need to recreated.

The adversarial node keeps track of the public chain, as well maintains a private chain for its own use.

In case the node is not honest, it does not broadcast the mined block unless specific conditions are met, such as when there is already a fork at the latest block of the blockchain, with one block created by an honest node and the other by the dishonest node. In this case, whoever mines the next block "wins" the game (and the associated reward). The dishonest node always mines at the end of its private chain, is there is one, or at the end of the longest existing public chain.

4 Termination

The simulation runs until the time t_t, given by $t_{max} + 2m$, where t_{max} is the time of the last transaction and m is the average mining time. This gives the network enough time to create a block which contains the last transaction, but terminates it not very long after. It has been empirically verified that only mining events are scheduled after this in the queue.

At the end of the simulation, the information regarding which nodes have which blocks is written to a single file, which is named as per the parameters of the simulation.

5 Conclusions

For the purpose of the following results, all combinations of the following parameters have been used:

Number of honest nodes, n: 100 Number of transactions, t: 50 Interarrival mean, i: 10 Average Mining time, m: 200 Fraction of nodes connected to the adversary, z: 0.10, 0.30, 0.50, 0.70, 0.90

ζ	MUP_{adv}	$MUP_{overall}$
0.1	0.0	0.5827814569536424
0.3	0.2	0.5550561797752809
0.5	0.5	0.5612472160356348
0.7	0.5714285714285714	0.5682819383259912
0.9	0.75	0.5773672055427251

Table 1: MUP of selfish mining attack

ζ	MUP_{adv}	$MUP_{overall}$
0.1	0.0	0.5827814569536424
0.3	0.2	0.5550561797752809
0.5	0.5	0.5612472160356348
0.7	0.5714285714285714	0.5502092050209205
0.9	0.6	0.5733041575492341

Table 2: MUP of stubborn mining attack

As can be seen from the tables 1 and 2, the work done by the adversary start to become profitable only after it is connected to at least half the nodes in the network, and this profit goes on increasing as the value of ζ increases.

Fig 1 shows a typical blockchain tree, created in a network of 11 nodes, wherein there are a few small chains that are abandoned soon after. The red blobs represent blocks created by the adversary, while the blue represent blocks created by the honest block.

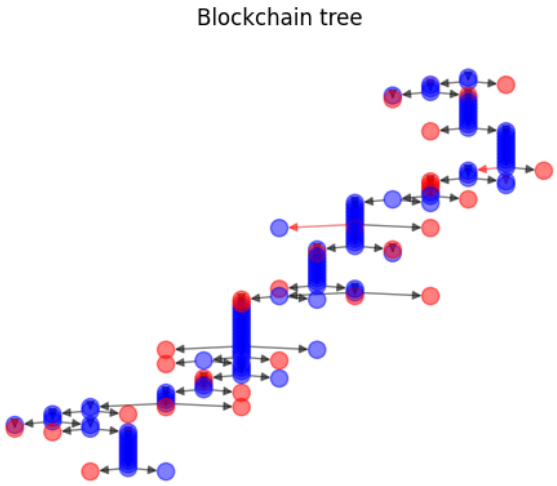

Figure 1: Typical Blockchain Tree

References

[1] I. Eyal and E. G. Sirer, "Majority is not enough: Bitcoin mining is vulnerable," *CoRR*, vol. abs/1311.0243, 2013.

[2] A. Baumann, B. Fabian, and M. Lischke, "Exploring the Bitcoin Network," vol. 1, Apr. 2014.

[3] L. Cocco and M. Marchesi, "Modeling and Simulation of the Economics of Mining in the Bitcoin Market," *PLOS ONE*, vol. 11, p. e0164603, Oct. 2016. Publisher: Public Library of Science.

[4] M. Javarone and C. Wright, "From Bitcoin to Bitcoin Cash: a network analysis," pp. 77–81, June 2018.

[5] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *science*, vol. 286, no. 5439, pp. 509–512, 1999. ISBN: 0036-8075 Publisher: American Association for the Advancement of Science.