Simulating Transaction Conflicts in the IOTA Tangle

Anya Zhang

CS 236R, Spring 2019

1. Introduction

Bitcoin, the most well-known and successful example of a decentralized digital currency, was launched in 2009. Its core innovation is the blockchain, a public ledger that provides a permanent and unequivocal record of bitcoin transactions. Since its debut, a multitude of other cryptocurrencies, such as Ethereum, Litecoin, and Zcash, have attempted to improve on various aspects of Bitcoin. In particular, many recent blockchain innovations focus on scalability, one of the most significant remaining barriers to the adoption of cryptocurrencies for commercial purposes. These include transacting through off-ledger peer-to-peer channels (Lightning and Interledger), additional layers of child chains (Plasma), as well as various forms of block compression. However, all of these come with tradeoffs of their own, including security concerns and difficulty of adoption.

Another solution, suggested by Lewenberg et al, proposes to allow even conflicting blocks to be published by changing the blockchain data structure from a linked list to a directed acyclic graph [1]. Essentially, the block DAG decouples the block publication step from the verification step. While classic blockchain requires that any block with a single conflicting transaction with the main ledger be rejected, resulting in tons of wasted computation, the DAG allows multiple blocks to extend the same parent block even if they conflict. In order to convert the DAG into a global ledger, the topological order of the DAG is ultimately used to select a main chain and incorporate off-chain transactions, as long as they do not conflict with previously included transactions. According to the authors of [1], this protocol results in a higher throughput and less discrimination against smaller players while maintaining the same level of security.

The most well-known implementation of DAGs is the protocol for the IOTA cryptocurrency, which was developed for communication and microtransactions between Internet-of-Things devices [2]. IOTA's DAG, called the "Tangle", includes several significant deviations from the original DAG proposal, most notably including scrapping the block concept entirely in favor of individual transactions and thus getting rid of both transaction fees

and block miners. The IOTA whitepaper describes the mathematical foundations of the tangle, with a focus on the MCMC algorithm it uses both to select the attachment sites for a new transaction (called a "tip") and to ultimately decide between conflicting transactions.

There currently exist several simulations of IOTA, including a 2018 Python codebase developed by Manual Zander [3] as well as the official JavaScript simulation published by the IOTA foundation [4]. However, both model how the Tangle grows without taking into consideration the data contained within transactions, despite the fact that the main innovation of the Tangle is to allow the publication and delayed verifications of conflicting transactions. Thus, the main focus of this project is extending simulation [3] to include conflicts, and studying the results thereof.

The main contributions of this project are the following:

1. Creation of a model for and extension of [3] to simulate conflicting transactions, which no existing simulations currently support.

2. Modification of [3] to support agents playing different strategies, including new selfish and lazy strategies.

3. Simulation of lazy and selfish strategies showing that when conflicts are included, both become much more effective against the default MCMC strategy.

2. Background: The Tangle

2.1. Definitions

In the tangle, a DAG of transactions, rather than a linked list of blocks, becomes the ledger. Each incoming transaction has to approve exactly 2 transactions, represented by a directed edge from the approver to the approved. If there is no directed edge between transactions a and b, but a path from a to b exists, a indirectly approves b, similarly to the relationship between blocks in the blockchain. All of the transactions lead back to the "genesis" transaction. Thus, the users or nodes in the network use new transactions to approve prior transactions and ensure that there are no conflicts in any lines of descent back to the genesis. Although the tangle does not attempt to create a single history, as transactions receive approvals, they have an increasing probability of being deemed "confirmed".

The parameters controlling the shape and evolution of the tangle are the following:

- λ, the rate of the Poisson process modeling transaction arrivals
- α, the determiner of how much the tip selection algorithm (discussed below) favors heavier transactions
- k, the number of tips approved by each incoming transaction, set to $k = 2$
- N, used in this paper to represent the total number of transactions simulated
- h, an approximator of the network latency (minimum time between publication and approval of a transaction), usually set to $h = 1$

2.2. Tip Selection Algorithms

Unapproved transactions (at the perimeter of the tangle) are referred to as "tips", and the main focus of [2] is the discussion of various algorithms for choosing which tips to approve. Before getting into security questions, the major consideration is lazy approvals, which refers to the strategy of approving extremely old tips instead of staying up to date with the network. Because lazy approvals contribute zero growth to the network, they must be discouraged through tip selection algorithms. For example, the most basic algorithm is selecting uniformly at random between existing tips to approve. However, this method is not robust against lazy behavior, because a tip that lazily approves other old tips is equally as likely as a useful tip to be chosen by the next transaction. Another algorithm is the unweighted random walk, which consists of letting a walker begin at the genesis and travel up the tangle to a tip, choosing uniformly at random at any fork. This algorithm has the benefit of prioritizing better-connected tips, but still fails to discourage lazy behavior.

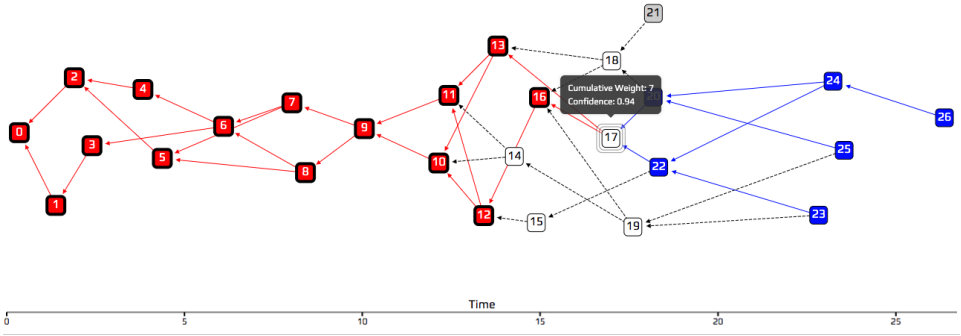
Instead, IOTA uses an algorithm based on Markov Chain Monte Carlo (MCMC), which on top of solving the problem at hand, can also be adapted to solve a number of security issues in the tangle. Under this algorithm, a random walker also begins at the genesis and moves towards the tips, but chooses which path to take from a given node based on the cumulative weight of each of its predecessors, where the cumulative weight $\mathcal{H}_x$ of a transaction x is calculated as the number of its direct and indirect approvers + 1. The transition probability of for this walk is then given by:

$$P_{xy} = \frac{e^{-\alpha(\mathcal{H}_x - \mathcal{H}_y)}}{\sum_{z:z\sim x} e^{-\alpha(\mathcal{H}_x - \mathcal{H}_z)}}.$$

Using this method of tip selection, any lazy tip would be extremely likely to be passed over by the algorithm, because old transactions have many other much better-connected (heavier) approvers.

The same algorithm is also used to determine the confirmation confidence of any particular transaction. The tangle uses a frequentist MCMC approach, which conducts a large number (e.g. 100) of random walks from the genesis block, and assigns each transaction a probability of confirmation corresponding to the number of times they end up being indirectly approved by the final tip.

Figure 1: Example of a tangle with $\lambda = 0.8$, $\alpha = 0.5$, and $N = 25$. The highlighted transaction (transaction 17) has a cumulative weight of $6 + 1 = 7$ and confidence 0.94.



3. Simulating Transaction Conflicts

For this project, I extended an existing Python IOTA simulation framework developed by Manuel Zander at Imperial College London [3]. This framework focuses on the interactions between honest nodes (called agents), and includes support for interactions between multiple agents, network latencies, and all three tip selection algorithms described above. The main methods for both simulator classes (for single or multiple agent systems) are `setup()` and `run()`. `setup()` intializes instances of the `Agent` and `Transaction` classes with Expo(λ)-distributed interarrival times. `run()` iterates through the transactions, assigns each uniformly at random to an agent, determines which tips are visible to the agent, and selects two for approval using the new transaction based on the chosen tip selection algorithm. Upon finishing execution and creating the DAG, the framework offers many options for analysis and visualization of the results.

3.1. New Conflicts Model

However, one of the main shortcomings of the framework is a lack of support for conflicting transactions, which in addition to being a fundamental feature of the tangle are also required in order to model many types of security attacks. From a (informal) theoretical standpoint, our objective is to model transaction conflicts in such a way that satisfies the following property:

- Approval of any transaction a entails approval of every transaction approved of by a, and conflicting with any transaction a entails conflicting with every transaction that approves of a.

We may model this behavior by assigning each transaction a piece of "direct data" that is unique to that transaction, as well as a set of "indirect data" comprised of the direct data of each transaction of which it approves. Thus, any incoming transaction can only approve a tip if its direct data does not conflict with any of the tip's indirect data, and approval constitutes adding all of the tip's indirect data to its own set of indirect data. The model thus consists of (1) a rule for generating direct data for each transaction and (2) a rule for deciding whether two transactions conflict. It must also be true that conflicts are structure-independent - i.e., two pieces of data that conflict will do so regardless of location, timing, or order in the DAG.

For (1), we model each piece of data as a unique integer, calculated by:

$$\{\text{transaction data}\} := n \times \{\text{transaction ID}\} + x,$$

where x is an integer chosen uniformly at random from the range $[0, n)$ for some n. We implement this equation in the `__init__` function of the `Transaction` class. Consequently, given n and a number generated using this formula, we can deduce both the original value of x and the original transaction ID.

Using this property for (2), given an incoming transaction a with direct data d_a and any candidate tip b with indirect data $D_b = \{d_b\}$, we define a conflict such that a conflicts with b if and only if $d_b \neq d_a$, but $d_b \equiv d_a \mod n$ for some d_b. In other words, a conflicts with b if there is any indirect data in b that happens to hash to the same x value as d_a, but was generated by a different transaction. This extends to any two transactions in the graph, not just an incoming transaction and a tip, if we consider the full set of indirect data $D_a = \{d_a\}$ for an arbitrary node a. We have thus created the desired quality of conflicts that occur randomly, but never between two pieces of data with the same source.

3.2. Updated MCMC Algorithm

We implement this definition in the `get_unconflicting_tips` function of the `Multi_Agent_Simulation` class. In pseudocode, the entire tip selection process for an incoming transaction using weighted MCMC is now:

Algorithm 1: Weighted MCMC Tip Selection

Input: Incoming transaction
Result: Selects 2 tips for approval using weighted random walk; updates DAG and transaction data
2 initialize `tip1` and `tip2` to null
3 get list of valid tips for transaction (i.e. unapproved and visible to transaction owner)
4 **for** *tip in valid tips* **do**
5 | if there is no data in tip such that tip data and transaction data have matching x-values and differing origin ids, add to list of unconflicting tips
6 **end**
7 conduct weighted random walk from genesis; stop when reach an unconflicting tip and set to `tip1`
8 indirect data of transaction ← union of itself and indirect data of `tip1`
9 update unconflicting tips to include new indirect transaction data
10 repeat lines 7-8 to get `tip_2`
11 **if** *tip1 and tip2 both non-null* **then**
12 | add edge between transaction and `tip1` and between transaction and `tip2`
13 **end**

An important note about this algorithm concerns line 11 - rather than connecting whichever of the two tips does not conflict with our new transaction, we throw out the transaction entirely (in the visualization, it becomes orphaned) if we can't approve exactly two tips. This will prove crucial for our later results.

3.3. Analysis of Parameters

We observe that under this model, as later transactions accumulate more and more indirect data from their descendants, it becomes more and more difficult over time to find two tips that don't conflict with an incoming transaction, or which is more often the problem, with each other. Given two transactions that each have depth h (defined as the maximum length over all paths to the genesis) and thus each have indirect data consisting

of at least h randomly chosen data points x in the range $[0, n)$, a simple combinatoric calculation tells us that the probability that there will be no overlap between the two data sets has an upper bound

$$\frac{\binom{n}{h} \times \binom{n-h}{h}}{\binom{n}{h}^2} = \frac{\binom{n-h}{h}}{\binom{n}{h}}.$$

According to [5], the expected number of tips at any given time is given by 2λ, assuming the default value of $k = 2$, which means that the expected total height of the Tangle will be $\mathbb{E}(H) = \frac{N}{2\lambda} = 12.5$ for default values $N = 50$, $\lambda = 2$. In this case, by the last few transactions, the probability of successfully approving two tips becomes:

$$\frac{\binom{n-h}{h}}{\binom{n}{h}} = \frac{\binom{n-12.5}{12.5}}{\binom{n}{12.5}},$$

which evaluates to $p \approx 0.015$ for $n = 50$ and $p \approx 0.168$ for $n = 100$. We've left this as another parameter that can be configured (via its inverse, `_conflict_coefficient` $= 1/n$, which is the probability of any two unconnected transactions conflicting.)

4. Agent Strategies

Another drawback of the original codebase was that it supported only a single global tip selection strategy, assuming that all agents would play in the same honest manner. By slightly modifying the attributes and initialization of the simulation and agent classes, we added support for individual agent tip selection algorithms. Additionally, this project adds two new algorithms, `lazy_selection()` and `selfish_selection()`.

4.1. Lazy Selection

The lazy selection algorithm has the same structure as the other tip selection algorithms, but given a selection of valid and unconflicting tips, chooses the two with the minimal amount of indirect data. We know this strategy fails when conflicts are not being considered - however, we include it because when conflicts are involved, a lazy tip has the lowest probability of conflicting with any future tip and thus could prove an attractive option for new transactions.

4.2. Selfish Selection

The selfish selection algorithm, when given a selection of tips, prioritizes ones with the same owner as the incoming transaction. This is consistent with selfish agents in other contexts, but hasn't been studied extensively in a DAG context. Intuitively, given that all other agents are cooperative, this should also be advantageous, because it increases the probability that other agents' transactions become orphaned. When conflicts are being considered, it has the additional advantage of reducing the amount of new/conflicting information that needs to be added to each transaction's indirect data with each subsequent approval.

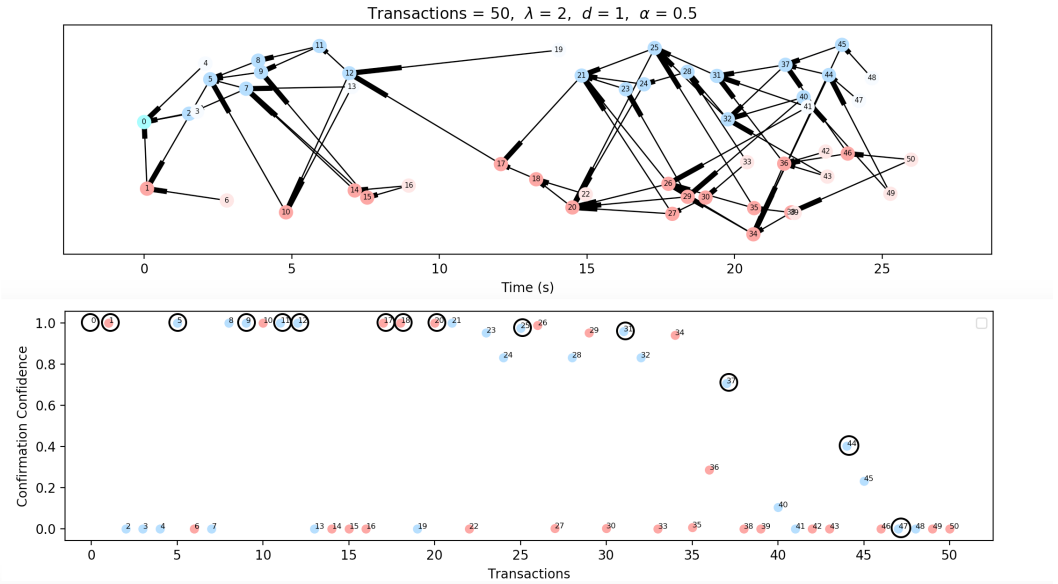
4.3. Comparing strategies

Finally, before being able to discuss the results of different agent strategies, we must establish the standards used for evaluating their relative success. As briefly mentioned earlier, according to the original DAG proposal, the main chain should be found through a time-based topological sort of transactions. According to the IOTA whitepaper, nodes are probabilitistically added to the main chain using the same weighted random walk from the tip selection algorithm and are considered confirmed with "confidence" when this probability exceeds a threshold value of $p = 0.95$. According to a subsequent study of the "Probability of being left behind" (POBLB) in Tangle v0.2 [6], the main chain can be decided according to the GHOST protocol, which deterministically adds the heaviest node at each point.

The simulation framework uses the probabilistic definition from the IOTA whitepaper, calculating the "exit probability" (probability of being reached) of each tip, so that each node's confirmation confidence is the sum of exit probabilities over all tips that are both visible and connected to it. Although confirmation confidence is a helpful measure for individual simulations and plots, the framework also provides an overall attachment probability for each agent, calculated by summing exit probabilities over each agent's tips. Intuitively, this is the probability for each agent that the next tip approved under MCMC will be one of their own. We use this metric to ultimately measure the success of each agent over a large number of trials.

Another of this project's contributions is the implementation of the deterministic definition as well through the function `get_heaviest_chain()`. However, we additionally observe that the heaviest chain is unsurprisingly strongly correlated to (i.e. almost exclusively drawn from) nodes confirmed with high confidence, so we focus on the above metrics throughout the rest of this section. (See Figure 2 for example).

Figure 2: Simulated tangle for 2 agents with MCMC tip selection. Top: DAG visualization with time on the X-axis. Blue transactions correspond to those assigned to Agent 0, and red to those assigned to Agent 1. Bottom: Confirmation confidence for each transaction. Transactions corresponding to the output of `get_heaviest_chain()` are circled in black. Note the correlation between confirmation confidence and circling.



5. Results

5.1. Partial Success of Lazy Behavior

We first compute several baseline metrics to test how well our simulations correspond to theoretical predictions. By averaging over 100 trials, we see that before introducing conflicts, the weighted agent has an overall attachment probability of 99.88%, while that of the lazy agent is only 0.1204%, as expected. Additionally, after enabling conflicts, pitting two weighted agents against each other for 100 trials results in an even split of 49.19% and 50.81%.

However, when we pit a weighted agent against a lazy agent, we obtain results that mirror our earlier intuition - Figure B.3 suggests a clear trend that the higher the conflict probability, the better lazy agents do against weighted agents, to the point that they are almost competitive. For higher values of the conflict probability corresponding to values of n around 2 or 3, this is a result of largely degenerate cases, where the vast majority of nodes are orphaned (see Figure B.4 for an example) as a result of the aforementioned

requirement that each transaction approve exactly 2 tips. Even for $n > 10$, however, this indicates a significant relationship that is worth exploring further both in theory and in simulation, as it has not been predicted in the literature at all.

5.2. Success of Selfish Behavior

For our selfish strategy, we test the effects of four combinations of parameters, consisting of every combination of 2 selfish agents or 1 selfish and 1 weighted agent with conflicts enabled or disabled. The results are summarized in the following table:

	weighted, selfish	selfish, selfish
conflicts off	[0.5877, 0.4123]	[0.4858, 0.5142]
conflicts on	[0.1151, 0.8849]	[0.5040, 0.4960]

Incredibly, though evenly matched or even slightly worse for all other three categories, a selfish agent when competing with a weighted agent with conflicts on outperforms them 88.49% of the time. We investigate the effects of this result for different values of n as well in Figure B.5. The explanation for this phenomenon, depicted in Figure B.6, also matches our earlier intuition - where the weighted agent has an increasingly large number of orphaned transactions, the selfish agent avoids orphaning by sticking to known branches, which have a much smaller pool of data and thus conflicts.

6. Conclusion

This project contributes major extensions to the multi-agent IOTA simulator developed by Manuel Zander. It uses the addition of both a theoretical model and simulation framework for conflicts as well as new agent strategies, all of which have been largely or completely overlooked in current research. Using these, we see that when conflicts are likely, lazy agents' probability of success increases from negligible to comparable, while that of selfish agents increases from comparable to dominating. Additionally, it improves upon many other features of the codebase, including methods for calculating the weight of any given transaction, deterministically generating the heaviest chain, and plotting the confirmation confidence of color-coded transactions. This model for conflicts is quite simplistic and perhaps excessively random - in future studies, it may be more realistic to consider a model that generates conflicts deliberately, in a way such that individual agents at least

are internally consistent. Simulating attacks such as those described in the IOTA whitepaper, in particular, require malicious agents that are able to insert specific conflicts in tangle history. However, these results still represent an interesting first point of discussion, and demonstrate that considering conflicts and their properties is an important part of research on the tangle.

References

[1] Y. Lewenberg, Y. Sompolinsky, A. Zohar. Inclusive Block Chain Protocols. *Financial Cryptography*, 2015.

[2] S. Popov. The Tangle, 2016.

[3] M. Zander, T. Waite, D. Harz. DAGsim: Simulation of DAG-Based Distributed Ledger Protocols. *ACM SIGMETRICS Performance Evaluation Review*, 46(3)118-121, 2018.

[4] A. Gal. The Tangle: an Illustrated Introduction. *IOTA Foundation*, 2018.

[5] B. Kusmierz, P. Staupe, A. Gal. Extracting Tangle Properties in Continuous Time via Large-Scale Simulations. *IOTA Foundation*, 2018.

[6] B. Kusmierz, A. Gal. Probability of being left behind and probability of becoming permanent tip in the Tangle v0.2. *IOTA Foundation*, 2018.

Appendix A. Code

Github repo: https://github.com/anyazz/iota_simulation

Appendix B. Figures from Results

Figure B.3: Attachment probability of lazy agent (as opposed to weighted agent) as conflict probability $= 1/n$ increases, along with polynomial best fit line. It appears that there is a theoretical limit preventing the lazy agent from reaching 50%.

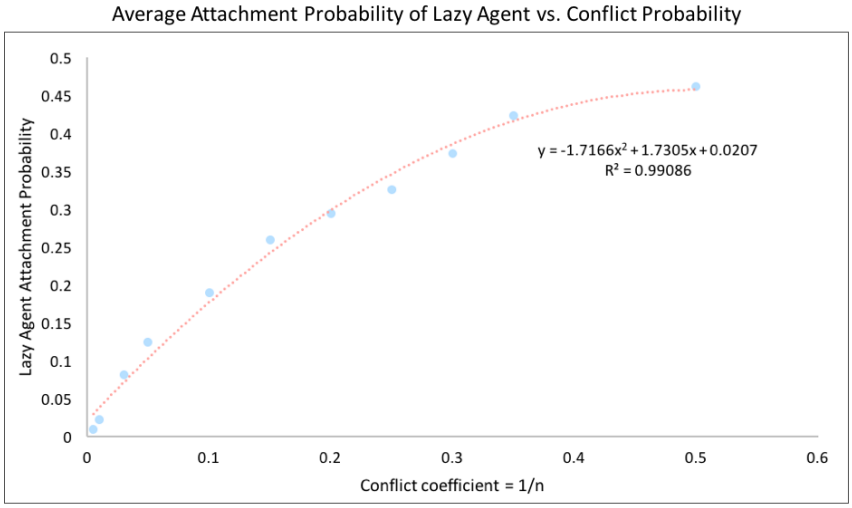


Figure B.4: Simulated tangle for weighted agent (blue) vs. lazy agent (red). Top: DAG visualization with time on the X-axis. Note the large number of orphaned blue transactions. Bottom: Confirmation confidence for each transaction.

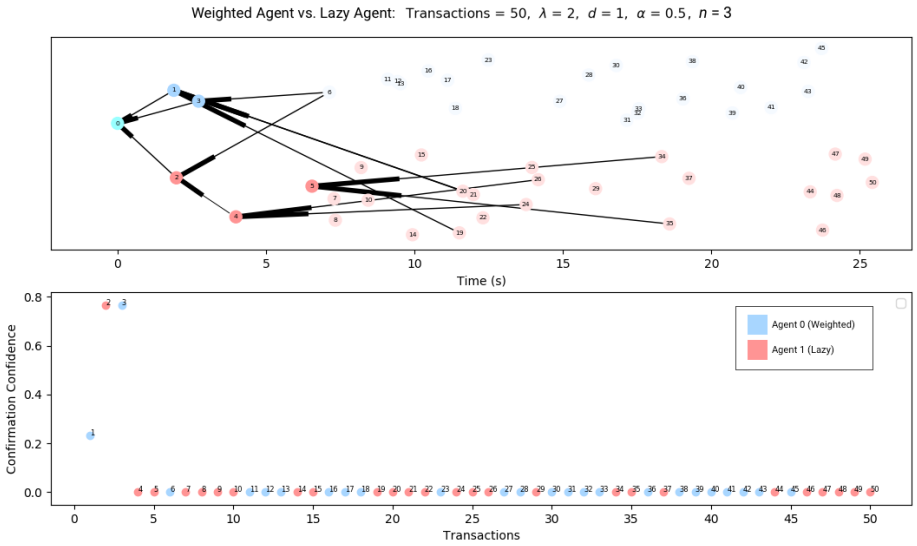


Figure B.5: Attachment probability of selfish agent (as opposed to weighted agent) as conflict probability $= 1/n$ increases, along with polynomial best fit line. It appears that there is a theoretical peak in success rate around $n = 4$ or 5.

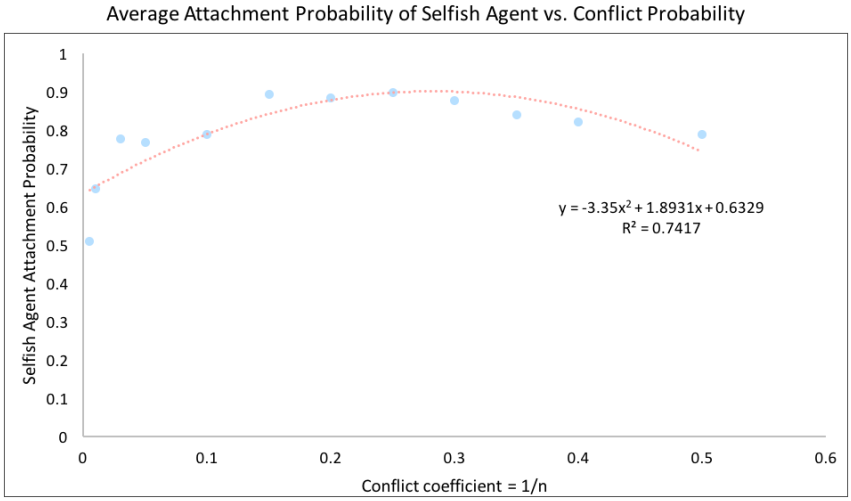


Figure B.6: Simulated tangle for selfish agent (blue) vs. weighted agent (red). Top: DAG visualization with time on the X-axis. Note the large number of orphaned red transactions. Bottom: Confirmation confidence for each transaction.