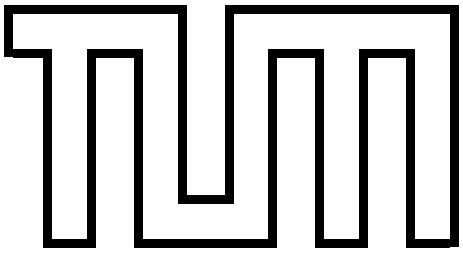


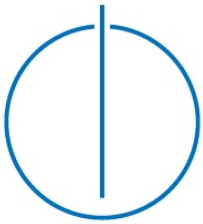
Technische Universität München

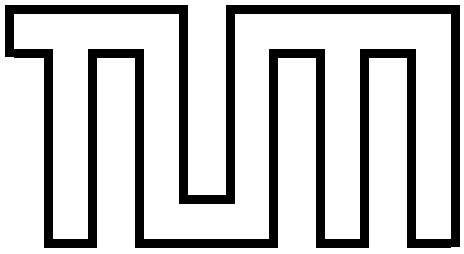
Fakultät für Informatik

Master's Thesis in Informatik

Design and Implementation of a Simulation Framework for Blockchain Alternatives: A Directed Acyclic Graph Based Distributed Ledger

Mohamed Riswan Abdul Lathif





Technische Universität München

Fakultät für Informatik

Master's Thesis in Informatik

Design and Implementation of a Simulation Framework for Blockchain Alternatives: A Directed Acyclic Graph Based Distributed Ledger

Entwurf und Implementierung eines Simulationsframeworks für Blockchain-Alternativen: Ein gerichteter azyklischer graphbasierter Distributed Ledger

Author: Mohamed Riswan Abdul Lathif

Supervisor: Prof. Dr. Hans-Arno Jacobsen

Advisor: Pezhman Nasirifard

Submission: 15.01.2019

I confirm that this master's thesis is my own work and I have documented all sources and material used.

München, 15.01.2019

(Mohamed Riswan Abdul Lathif)

Abstract

Blockchain based distributed ledgers is one of the most studied research areas, owing to the enormous success of crypto-currencies like Bitcoin, Etherium, etc. But, in addition to all the advantages, Blockchain is by design resource intensive and in general does not scale well. Hence, researchers all over are looking for alternatives for blockchain and one such alternative is the Tangle, introduced by IOTA foundation. Tangle is a Directed Acyclic Graph (DAG) based distributed ledger, which boasts advantages like high scalability and support for micro-payments. But outside the context of IOTA, the properties of a DAG-based distributed ledger are not studied. Also IOTA currently does not contain a large scale peer-to-peer simulation system with configurable parameters, which allows the users to study the important characteristics and metrics of the network and compare different scenarios under controlled conditions. This thesis proposes CIDDS, a Configurable and Interactive DAG based Distributed ledger Simulation framework as a solution to this problem. Using CIDDS, users can create large scale tangle simulations with thousands of nodes and study the characteristics of the resulting DAG ledgers with varying parameters.

Inhaltsangabe

Blockchain basierte Distributed Ledgers gehören in der heutigen Zeit zu den größten Forschungsfeldern. Die Gründe liegen hier besonders in den Erfolgen von Kryptowährungen wie Bitcoin, Etherium, usw. Trotz vieler Vorteile sind Blockchain Systeme meist ressourcenintensiv und schwer skalierbar. Daher suchen viele Wissenschaftler nach Alternativen für Blockchain Systeme. Eine dieser Alternativen ist Tangle, welches von der IOTA Stiftung ins Leben gerufen wurde. Tangle ist ein auf gerichteten zyklischen Graphen (Directed Acyclic Graph oder DAG) basierendes Distributed Ledger System, welches Vorteile wie etwa hohe Skalierbarkeit oder Unterstützung für Micropayments fördert. Außerhalb des Kontextes der IOTA Stiftung wurden die Eigenschaften von DAG basierten Distributed Ledger Systemen allerdings noch nicht erforscht. Des Weiteren enthält IOTA bislang noch keine großen Peer-To-Peer Simulationssysteme mit anpassbaren Parametern, welche den Nutzern das Erforschen von wichtigen Charakteristiken bzw. Metriken des Netzwerks und das Vergleichen von verschiedenen Situationen in kontrollierten Bedingungen ermöglicht. Als Lösung dieses Problems schlagen wir CIDDS, ein konfigurierbares und interaktives DAG basiertes Distributed Ledger Simulationssystem (eng. Configurable and Interactive DAG Based Distributed Ledger Simulations Framework), vor . Mit CIDDS können Nutzer eine große skalierbare Simulation mit tausenden von Knoten erstellen und die Eigenschaften der entstandenen DAG Ledgers mit unterschiedlichen Parametern erforschen.

Acknowledgment

I would like to extend my sincere thanks to my advisor Pezhman Nasirifard for being the best advisor possible and helping me out with all the issues I faced and steering me in the right direction. I would also like to express my gratitude for my supervisor Prof. Dr. Hans-Arno Jaconbsen for providing me an opportunity to write this thesis at his chair at TU Munich and for the constant inspiration.

I would also like to thank Alon Gal from the IOTA foundation for his valuable inputs in designing this simulator. Also I express my gratitude to Minh-Nghia Nguyen for his correspondence and input in implementation of the tip selection algorithms used in the simulator.

I am grateful for all my friends and family, especially my mother who has been a constant source of motivation throughout the duration of the thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

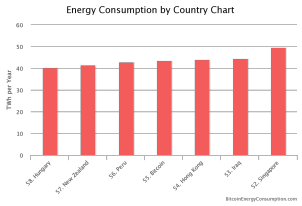
Blockchain based distributed ledgers have taken the computing world by storm and the huge success of Bitcoin and other cryptocurrencies has sparked an academic interest around the Distributed Ledger Technology (DLT). Unfortunately, Blockchain based DLTs also have major disadvantages which cannot be overlooked and hence extensive research towards alternatives for Blockchain based DLTs is needed. One such alternative is the Directed Acyclic Graph (DAG) based DLT.

DAG based DLTs have gained prominence when the IOTA foundation introduced the Tangle [1] a DLT which can scale infinitely, at least in theory. This claim warrants a deeper investigation and study as this would be a step in right direction to mitigate the issues with Blockchain based DLTs.

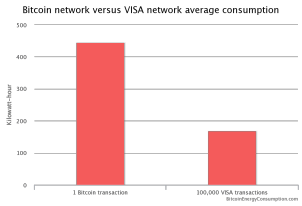
1.1 Motivation

There are major disadvantages with Blockchain based DLT solutions like Bitcoin [2], Etherium[3], etc. Some of the major issues are:

1. **Scalability**: Blockchain based systems suffer a major bottleneck with scalability. In Bitcoin the average transaction time is around 1 hour and Etherium has an average of 15 approved transactions per second. This would make these systems almost impossible for real time consumer based transactions like grocery shopping.
2. **Impact on Environment**: The Proof of Work (PoW) or Proof of Stake (PoS) models of block creation waste energy and this has major impact on use of resources like electricity.



(a) Bitcoin Energy consumption



(b) Bitcoin Carbon Footprint

Figure 1.1: Bitcoin - Impact on the Environment

Figure 1.1a shows that if Bitcoin were to be a country, it would be the 55th in the world in energy consumption at the time of writing this thesis. Figure 1.1b shows that 1 Bitcoin transaction has almost 3 times higher carbon footprint when compared against 100,000 Visa transactions[4]. The DAG based alternative would essentially reduce the computing required for approval of a transaction and could reduce the negative impact on the environment.

3. **No support for Micro-transactions**: Microtransactions require a reward system that does not rely too much on transaction fees. This is hard to implement in a blockchain based system when mining reward is taken away.

4. **Complex and Slow network**: Blockchain based systems suffer from high latency [5]. The complex underlying network also deters casual adopters from switching over to distributed ledgers.

All these factors and more serve as motivators for researchers to explore alternatives for Blockchain based DLTs. There are many simulators available for Blockchain based DLTs [6], [7]. Unfortunately next to no simulators are available in public domain to study the behaviour and characteristics of a DAG based DLT. This forms the motivation for the thesis.

1.2 Problem Statement

As of December 2018, the time of writing this thesis, there are a little more than 2000 available cryptocurrencies with a combined market capital of over 130 Billion USD [8]. This undeniable impact of DLT demands further studies into determining the future ramifications. A simulator could be a powerful tool in studying the underlying network properties of these DLTs with the user exerting varying degree of control over the network parameters. As outlined above, a simulator which can closely recreate a DAG based DLT would be a very useful tool for both researchers and students who would like to study the behavior and properties under controlled conditions.

The properties which the user would be able to modify are the following:

1. **Number of Processes**: These relate to the number of IOT devices which issue the transactions
2. **Number of Transactions** (n): The transactions that form the DAG
3. **Transaction Rate** (λ): The average number of incoming transactions per unit time
4. **Level of Randomness** (α): For a uniform tip selection with an added randomness
5. **Tip Selection Algorithm**: Uniform Random Tip Section (URTS) and Markov Chain Monte Carlo Tip selection (MCMC)

These parameters and their implications would be discussed in detail in the next chapter 4.

We have defined the following metrics for evaluation of the simulator:

1. **Correctness**: How close the results simulation fare with respect to the expected outcome.
2. **Speed**: How the simulator behaves while running small, medium and large-scale simulations.
3. **Scalability**: How well the simulator scales when the number of nodes and transactions is increased.
4. **Extensibiliy**: How easy it is to extend the simulator to include other configurable parameters or a whole different DAG framework like Hashgraph.
5. **Visuals**: Ability of the simulator to present the results in a clear, usable format.
6. **Comparison with IOTA's simulation results**: Comparing the result of CIDDS simulation against some of the IOTA's published simulations [9].

The evaluation results are discussed in detail in chapter 5.

1.3 Approach

This thesis aims to create a simulator which closely resembles the DAG proposed by IOTA in their whitepaper [1], while also giving room for extending it to support other DAG based distributed ledgers such as the Hedera Hashgraph [10]. This simulation framework is dubbed as Configurable and Interactive DAG based Distributed ledger Simulation framework (CIDDS).

One of the major focus would be to make it easy for the users to use the simulator with simple User Interface (UI), which also allows for comparing multiple simulation results. The major contrast with the similar simulators (detailed in chapter 3) would be the scale on which

CIDDS operates. Huge DAGs with more than 200,000+ nodes could be generated with varying parameters and studied with the help of CIDDS.

To achieve the scale and also model the DAG to simulate a realistic network, the transactions are modelled on the basis of Poisson Point Process and parallel processing is used to speed up the generation of the DAG and a process orchestrator is used to co-ordinate the entire process and resolve conflicts. The UI is built using Django [11] framework, as this is a powerful web framework which allows rapid building of large scale applications also allowing extensibility.

1.4 Contribution

The important contribution of this thesis is the attempt to create a complete simulation framework which can handle the scale needed by a DAG distributed ledger. To achieve this, parallelization of the simulation is essential to have the DAGs created in a reasonable speed. Also, the various tip selection algorithms are implemented, modelling a Poisson Point Process to achieve close to real world simulation results.

1.5 Organization

The general structure of thesis is presented here. The upcoming chapter introduces and briefly discusses all the necessary concepts to understand the thesis. Chapter 3 outlines the similar simulators and other works which are available currently and how these form the basis on which CIDDS is built upon. Chapter 4 details the design decisions and implementation details of the simulation framework. Chapter 5 evaluates the simulation framework with respect to the evaluation metrics outlined above. In Chapter 6, the conclusions are presented and future enhancements are discussed.

Chapter 2

Background

This chapter outlines the core concepts of the distributed ledger technologies, which would be the basis for the further chapters discussed in the thesis. We would define what distributed ledger technologies are, different types of DLTs which are widely used, the core technological concepts behind these DLTs and the importance of DAG based DLTs. Then a brief overview of the IOTA Tangle is presented. These concepts are discussed alongside the need of having a simulated environment for validating the correctness and viability of these technologies.

2.1 Distributed Ledger Technology

Distributed Ledger technology refers to a digital ledger, where a record is added by the consensus amongst all the parties which are distributed across a given domain. In this case, multiple copies of the ledger are kept independently by the participating parties. This removes the need for having a trusted third party, but introduces the overhead of reaching a consensus. Also, this removes the vulnerability of having a single point of failure. A distributed ledger is not destroyed till the last surviving node cease to exist. Thus security is achieved by design and a DLT has high Byzanthine Fault Tolerance.

According to the publication [12], the core concepts which define a DLT are as follows:

1. **Immutability**:
   This means that a data or record added to the DLT cannot be modified after a consensus is reached. This record is preserved to the life of the ledger, and hence adds an inherent security against modification of the record.
2. **Disintermediation**:

All the nodes involved in the DLT could interact with each other directly without the need for any intermediary node. This means that with the DLT, the nodes can create direct transactions between each other upon a cryptcurrency or a digital asset. (e.g Bitcoin as a cryptocurrency and Crypto-kitties [13] as the digital asset)

3. **No Centralized control**:
   Only the consensus is needed for validation of the transactions and hence there is no need for a centralized third party. Thus the need for a governing structure or a trusted third party is eliminated by the DLT. Figure 2.1 [14] illustrates this.

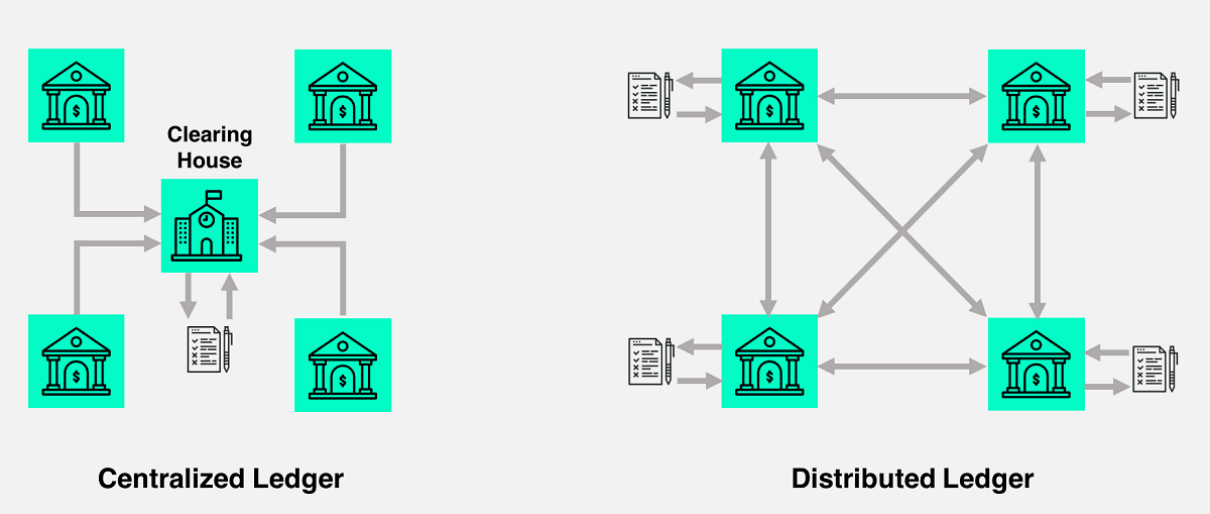


Figure 2.1: Centralized Ledger vs Distributed Ledger

4. **Novelty in storage and managing data**:
   Unlike traditional database based storage systems, DLT creates a need for exploration of novel methods of storage and distribution of data amongst the nodes.

The hype around DLT and the exploration for the possible uses of DLT started with the Bitcoin boom. This garnered huge attention towards the Blockchain based systems which would be discussed in detail in the next section.

2.2 Blockchain Based Systems

”A blockchain is essentially a distributed database of records or public ledger of all transactions or digital events that have been executed and shared among participating parties” [15]. In simple terms, blockchain is just a chain of blocks of transactions. Blockchain follows all the properties of a distributed ledger defined in the section 2.1.

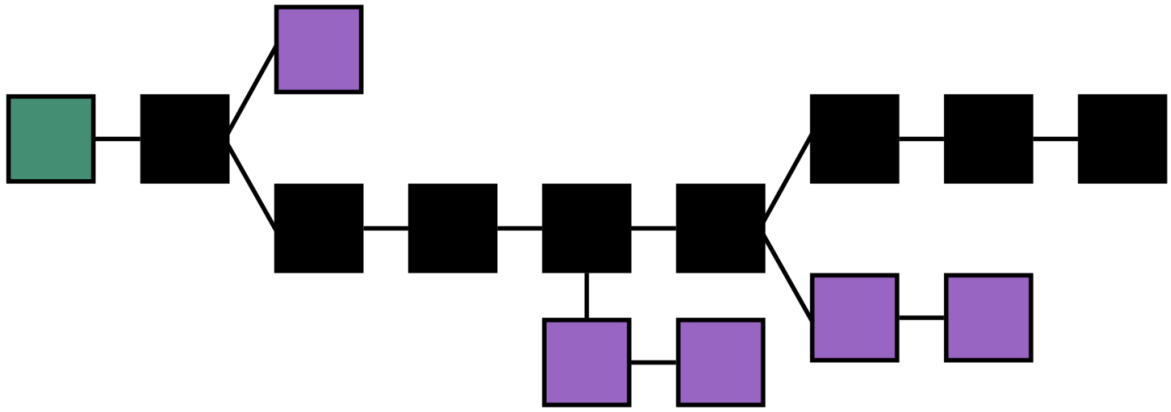


Figure 2.2: A typical Blockchain with side-chains

A typical blockchian with multiple side chains is referenced in fig. 2.2. In this case, the very first block, also referred as the genesis block is highlighted in green. The longest chain is shown in black and the side chains are shown in purple [16]. Usually in Blockchain the longest chain survives as it is vital to have a consensus. Though blockchain is often used synonymous with Bitcoin, Bitcoin would be discussed in subsection 2.2.3. In the following section, all the technological concepts which are necessary to understand a blockchain based system (and in extension a DAG based DLT) are outlined.

2.2.1 Technology behind Blockchain Based Systems

The initial blockchain based system was created from a proposal by an anonymous person or organization named Satoshi Nakamoto in the paper titled ”Bitcoin: A Peer-to-Peer Electronic Cash System”[2]. Blockchain based systems are heavily influenced by both Distributed systems and cryptography. Some of the core concepts needed to understand the thesis in detail are discussed briefly below.

P2P Networks

P2P networks are peer-to-peer networks. A formal definition from [17] is given as follows:

”A distributed network architecture may be called a Peer-to-Peer(P-to-P, P2P) network, if the participants share a part of their own hardware resources (processing power, storage capacity, network link capacity, printers). These shared resources are necessary to provide the Service and content offered by the network.”

Cryptography

Cryptography is the study of securing communication from the adversaries. It deals with creation of protocols and methodologies which prevents adversaries from reading unauthorized communication. Cryptographic principles, especially public key cryptography is vital in the working of distributed ledgers. Two terminologies, which are used widely in cryptography are: encryption and decryption.

Encryption

Encryption is a process of encoding a message such that adversaries do not understand or obtain the information from the encoded message. The encrypted message is often called as cyphertext.

Decryption

Decryption is the reverse process of encryption where the encoded message (the result of encryption) is reverted to its original readable form by authorized parties. This is generally the process of obtaining plaintext message from cyphertext.

Public Key Cryptography

Also known as asymmetric cryptography is a cryptographic system which uses two keys: a public key - which is published to the world and a private key - which is known only to the owner. In this case, anyone could encrypt the message using the public key but only the person with the private key could decrypt the message. This serves two distinct purposes:

- **Authentication** : The message sender can encrypt the message using the private key (also known as signing with a digital signature), which then can be decrypted using the public key. This could verify that the person who signed the message is indeed the sender, making authentication possible.
- **Non-repudiation** : Non-repudiation is achieved when no party can dispute the authorship of signing the message/document.

Cryptographic Hash function

A hash function which takes a message as an input and outputs a fixed length alphanumeric string. This output or hash value is otherwise known as checksum and digital fingerprint.

Gossiping

Gossip protocol is a Peer to Peer (P2P) communication where distributed systems pass messages to each other, sometimes even without relying on a central registry. It is based on how epidemics spread [18].

Consensus Protocols

Consensus protocols form a viral role in keeping the blockchain network synchronized. These are a set of rule which every participating node in a blockchain network follows to maintain an irrefutable agreement in the network, at the same time preventing any chance of exploitation by an adversary [19].

2.2.2 Bitcoin

Bitcoin is sometimes used synonymous with Blockchain, because Bitcoin whitepaper [2] was one of the first to bring out the applications of Blockchain. In general, blockchain is the technology which powers Bitcoin and many other systems (which are colloquially referred as ”Crypto systems” due to the heavy influence of cryptography). The timeline of this is shown briefly in fig. 2.3 [20]. We would briefly discuss the Blockchain based systems with reference to Bitcoin and then contrast it with other blockchain based systems.

Blocks

A typical Blockchain based system has blocks(which is usually a collection of transactions on a digital asset, but protocol can be modified to include any data), in which each block would refer to the previous block, usually by a pre-computed hash of the previous block. This is depicted in fig. 2.4 [20].

A new block can only be added to the system only if a node finds a solution to a difficult to solve mathematical problem (also known as PoW).

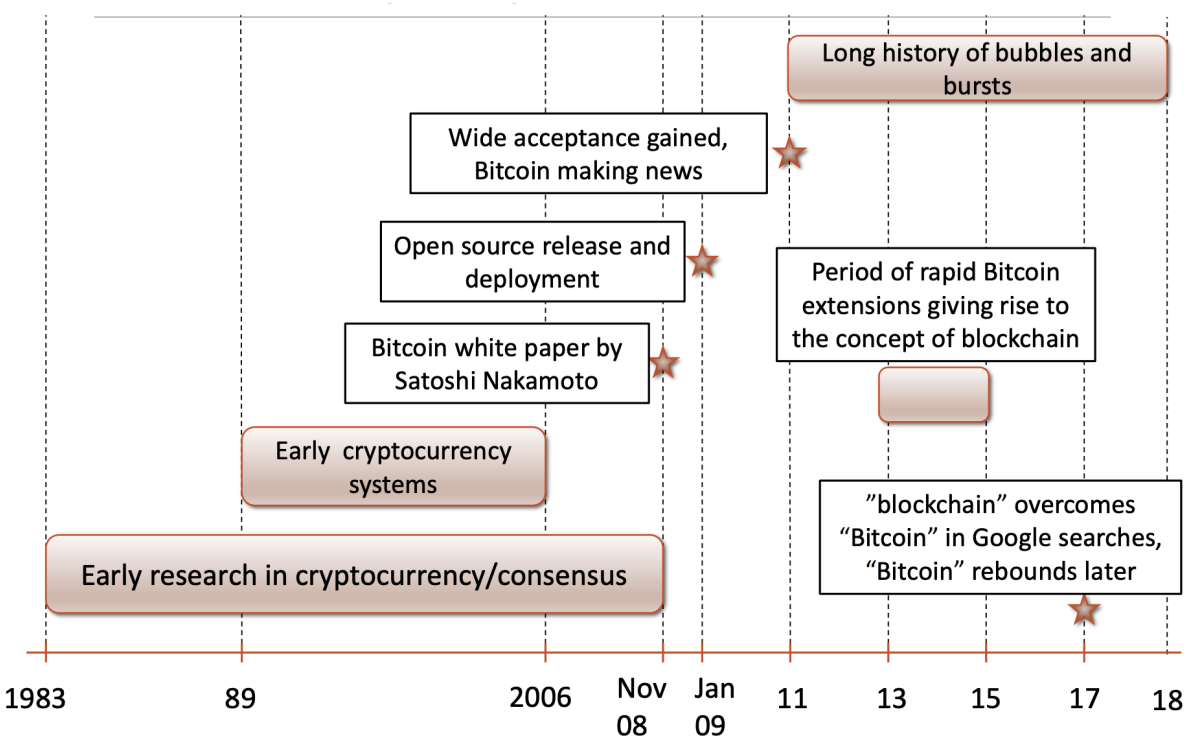


Figure 2.3: Blockchain based systems - A brief Timeline

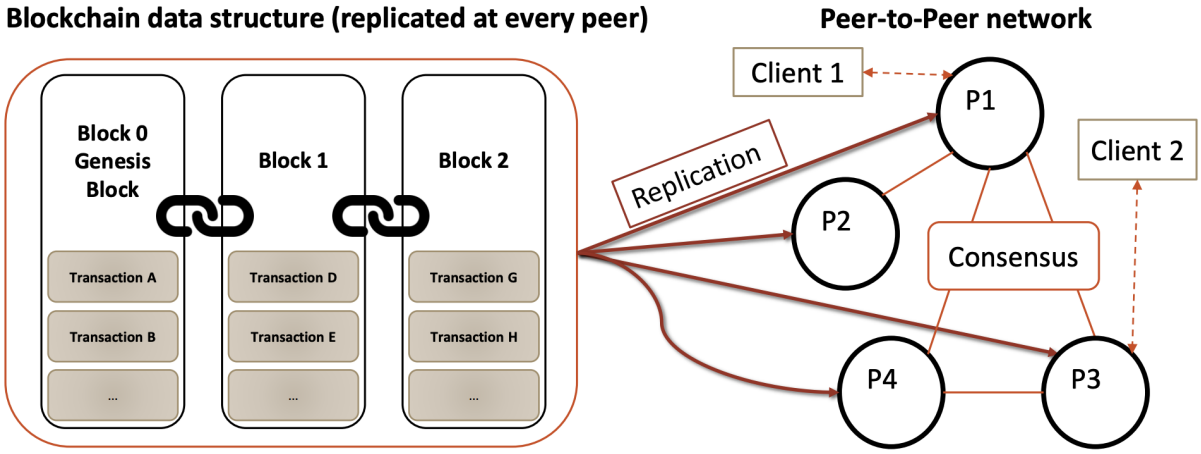


Figure 2.4: Blockchain Representation with Blockchain Network

Transactions

A transaction in a blockchain system transfers the underlying asset (in this case bitcoins) in the bitcoin network. Transaction can contain one or more inputs and outputs. One important thing to note is that the value of the input is always equal to the output.

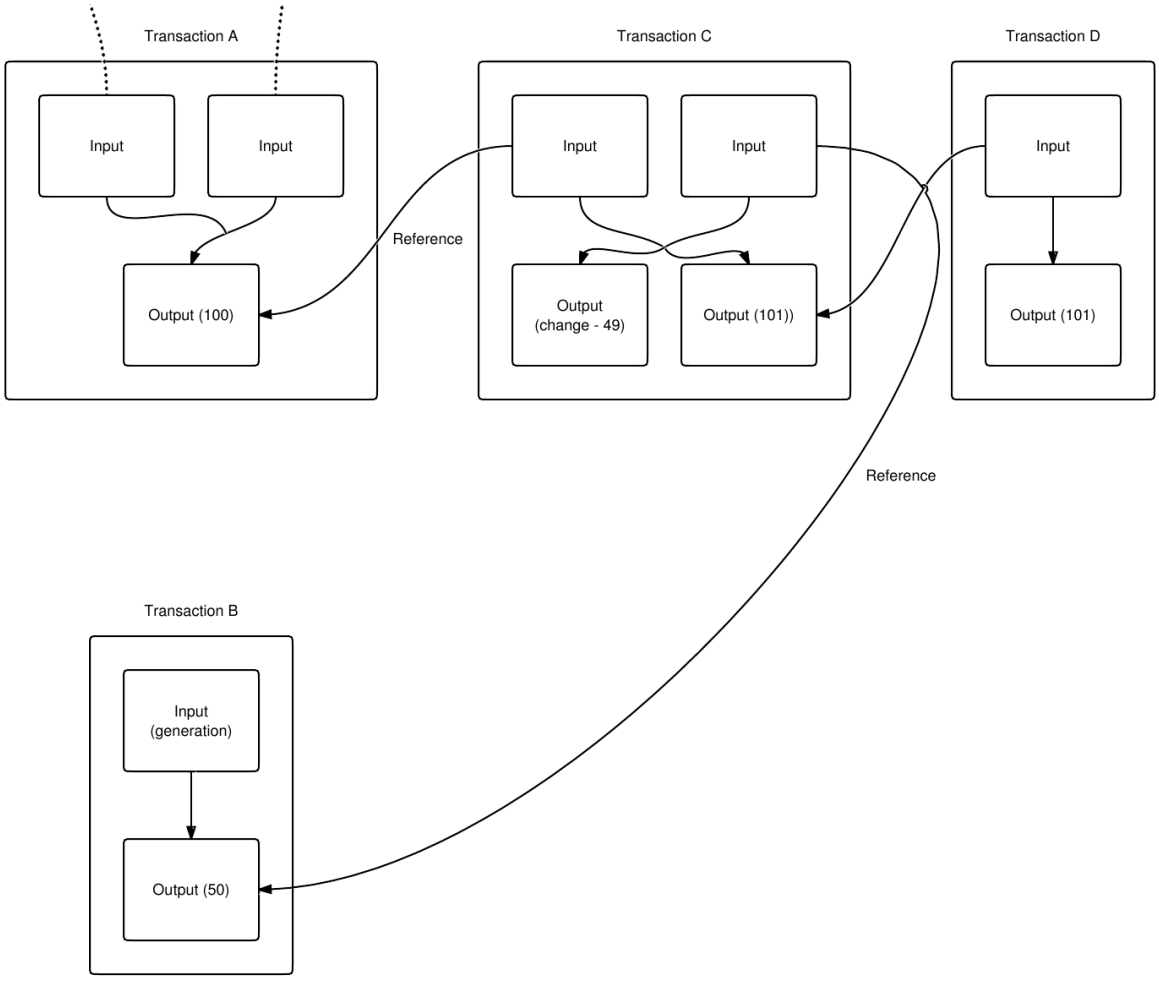


Figure 2.5: Blockchain Representation with Blockchain Network

A sends 100 BTC to C and C generates 50 BTC. C sends 101 BTC to D, and he needs to send himself some change. D sends the 101 BTC to someone else, but they haven't redeemed it yet. Only D's output and C's change are capable of being spent in the current state.

The process of sending and reciving Bitcoins via transaction is depicted in fig. 2.5 [21].

Though Bitcoin is one of the most prominent blockchain based system, this is not the only one. There are many more such systems, with a new one being proposed almost every other day. We would briefly discuss one other prominent blockchain based system, the Etherium.

2.2.3 Etherium

The Etherium blockchain, introduced by the Etherium foundation [3] is a platform for building decentralized applications. It runs smart contracts, or applications which run exactly as programmed on the blockchain without the possibility of a downtime.

One of the major difference from Blockchain based systems is the use of PoS instead of PoW for block creation. Also transactions done in etherium are faster.

Since this thesis deals with DAG based DLTs and not Blockchain based DLR, we would focus more on those.

2.3 DAG based Distributed Ledgers

Instead of using a Block, which is in most cases the bottleneck in a Blockchain system, a DAG based systems uses Directed Acyclic Graphs to represent the Distributed Ledgers. Instead of grouping the transactions into blocks and then linking the blocks, the DAG based DLT links the transactions themselves, thus overcoming the bottleneck of blockchain based systems.

In short each transaction, picks one or more transactions from the unapproved transaction pool and approves it. By this approval, this transaction becomes part of the network. But this introduces a lot of other complexities like achieving consensus, creation of protocols on selecting the unapproved transactions, etc. This is a very interesting topic of research and we outline the need of DAG based DLTs and the technology behind them in the upcoming sections.

2.3.1 Need for DAG based DLTs

Some of the major issues facing Blockchain systems and which can be mitigated by DAG based systems are:

1. **Serious Scalability Limitations**: With the current setup it is almost impossible for blockchain based systems to scale to a level to match the consumer market. Bitcoin network has a transaction average of 7 Transactions per Second (tps) while Visa network has around 2000 tps [20]. Compared to this a DAG based DLT has infinite scalability, at least in theory.
2. **High Transaction fees**: Currently, the reward for creation of a block is in the form of new bitcoins. But as the number of bitcoins is fixed, this number would become insignificant

over time. This means that the miners have to rely on transaction fees as incentive. This would likely increase the transaction fees in the future, which does not makes sense as it is exactly like having a trusted third party. This means that smaller transactions (also called microtransactions or micropayments) is impossible.

This is not the case with DAG based DLTs. Even devices with small computing power like IOT devices can issue transactions.

3. **Weakness of PoW**: The energy consumed for bitcoin block creation is atleast 1000x more than the use of credit card. Also the miner has a high bootstrap time [20]. Also the PoW process is slow, with block creation time set at around 8 minutes in average in Bitcoin network.

These are only some of the motivations for researching into an alternative for Blockchain based systems.

2.3.2 Technology behind DAG Based Systems

A DAG based system follows similar principles like a Blockchain system in terms of network propagation, but however differs vastly in terms of approval of transactions. Instead of being added to a block, the transactions are added to a directed acyclic graph, where they directly approve each other in the form of creation of links.

This thesis bases most of its work on IOTA's tangle, but the DAG is not referred as tangle because it can be extended to other DAG based DLTs as well.

2.3.3 IOTA Tangle

IOTA's Tangle is the largest DAG based Distributed Ledger while writing this thesis with a market cap of 872,022,608 USD [22]. Some of the major capabilities which IOTA boasts in contrast to Blockchain based systems are:

- **High Transaction Rates**: IOTA scales well, i.e as more and more nodes join the IOTA network, the transaction approval times reduce which result in very high transaction rates.
- **Support for Micropayments**: The core ideology of IOTA is making smaller IOT devices issue transactions without having to compute huge PoW. Hence smaller transactions also knows as Micropayments are possible.

- **Infinite Scalability**: Theoretically IOTA could scale infinitely as more nodes join the network. This is not the case with Blockchain based systems.
- **Future Readiness**: IOTA removes the bottlenecks in creation of Blocks. Also it is touted to be Quantum secure, i.e if and when quantum computers become mainstream, they could not break IOTA.
- **Immutability Assurance**: IOTA boasts of auditability by external applications.

The total supply of IOTA tokens is fixed at $(3^{33}-1)/2$ and hence it is a depreciating cryptocurrency as the tokens could be lost over time.

The tangle consists of transactions which form the vertices and the approvals which form the edges of the directed acyclic graph. If a transaction A approves transaction B which is already part of the IOTA network, then an edge A $\rightarrow$ B is created and added to the tangle.

To issue a transaction to the network a node must perform 5 operations:

1. **Creation of Transaction Bundle**: A bundle is the basic unit of transaction which contains the inpu, output and metadata. This step is also called as bundling and signing.
2. **Tip Selection**: In the context of IOTA, the tips are unapproved transactions in the IOTA network. Tip selection refers to selection mechanism of an unapproved transaction in the network.
3. **Validation**: This is the process of validating the transactions selected by the tip selection algorithm.
4. **PoW**: Unlike Blockchain based systems, the PoW is not as heavy in computation in the IOTA network
5. **Publishing**: This is the last step where the generated transactions are published to the network

2.3.4 Transactions in Tangle

A transaction is a node in the tangle. The anatomy of a transaction [23] is shown in fig. 2.6. Hash is the output of cryptographic hash function of all the other fields in the transaction. More detailed information of every single field can be obtained from the IOTA's documentation [24].

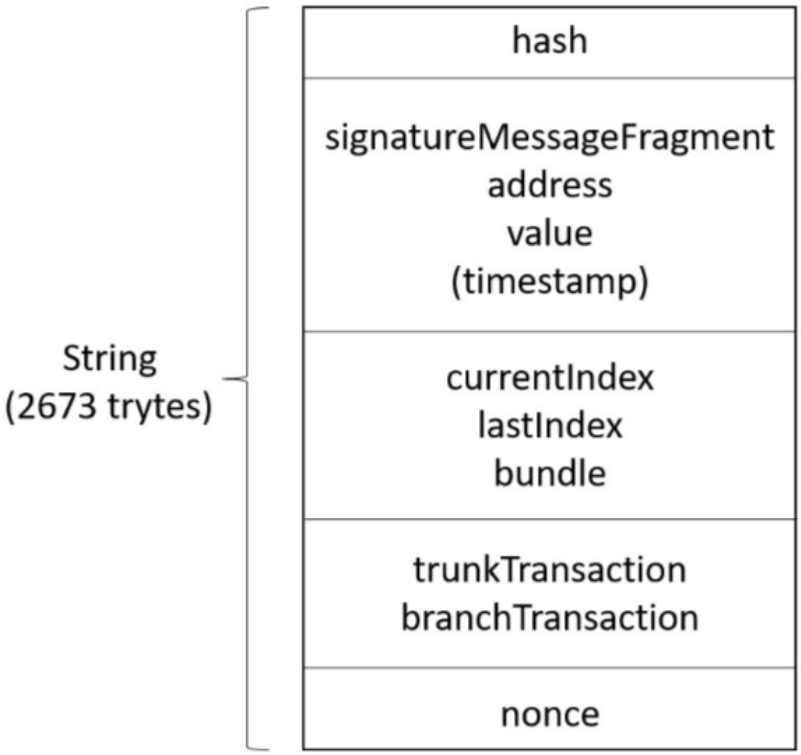


Figure 2.6: Structure of a transaction in Tangle

2.3.5 Tip Selection Algorithm

Another major aspect of the tangle which is very relevant for the this thesis is the tip selection algorithms used by IOTA. Two of the major tip selection strategies which are proposed by IOTA are:

- **URTS** : In URTS tip selection, the tips are selected randomly, i.e each tip has an equal probability to be selected among each other.
- **MCMC** : MCMC is an advanced tip selection strategy, where we assume there are some fixed number of particles, also known as walkers are placed in genesis or at a random node in the tangle and they ”walk” towards the tips based on a probability function. MCMC algorithm takes care that the tips are selected non-deterministically along the path of the largest cumulative weight in reasonable amount of time [23].

  To illustrate MCMC, lets assume we place a certain number of walkers N in the tangle either at the genesis or sufficiently deep. One possible heuristic to use is the set of transactions with cumulative weight between W and 2W. Another heuristic can be a set

of transactions arriving between a time interval $[t_0, 2t_0]$.

Once the heuristic is determined then each walker walks towards the tips. A walker can walk from node x to tip y iff $x \hookleftarrow y$, i.e there exist a path between x and y satisfying the given heuristic in the tangle $\mathbb{T}$. $x \leftsquigarrow y$ denotes y either directly or indirectly approves x. In this case, the transition probability between x and y (P_{xy}) can be calculated as follows:

$$P_{xy} = \frac{e^{-\alpha(H_x - H_y)}}{\sum_{z:x \leftsquigarrow z} e^{-\alpha(H_x - H_y)}} \quad x, y \in \mathbb{T}, x \hookleftarrow y$$

With the smaller difference between H_x and H_y, the walker loses less cumulative weight when it chooses y instead of any other transaction, lets say z. This density function is a valid Riemann density because the following properties hold true (we don't discuss the proof here, it can be referred from [23]):

1. $f \geqslant 0$
2. $\sum_{y:x \leftsquigarrow y} P_{xy} = 1$

Once the walker reaches the tips, the node is free to choose tip it wants to approve. Usually the tip which is reached first is selected.

These are some of the fundamentals of IOTA Tangle which would be used by this thesis.

2.3.6 Hedera Hashgraph

Hedera Hashgraph is a public permissioned blockchain built based on the virtual-voting consensus algorithm, invented by Dr. Leemon Baird [10]. It is fast because of the current permissioned settings, but it aims to become permissionless soon. It has an open consensus with a closed governance model. According to Hashgraph's website, hashgraph boasts the following properties:

- **Fast**: More than 250,000 tps
- **Fair**: Mathematically-Proven Fairness (via Consensus Time Stamping)
- **Secure**: Bank-Grade Security (by employing Asynchronous Byzantine Fault Tolerant)

This thesis would not focus on Hashgraph per se, but in theory it is possible to extend CIDDS to include Hashgraph based simulations.

2.4 Miscellaneous Mentions

2.4.1 Emulation

Emulation is the act of replicating the exact behavior of the real system, but in another environment.In other words, Emulation is the use of an application program or device to imitate the behavior of another program or device [25]. An emulated system must adhere to all the rules of the real system.

2.4.2 Simulation

A simulation is an approximate imitation of the operation of a process or system [26]. For simulation a simulation model or a framework is required. This model makes certain assumption about the real systems. In essence, only the needed aspects of the real system is captured in a simulated system. This thesis would focus on the simulation framework specifically.

Chapter 3

Related Work

3.1 Overview

For this chapter, we review the simulators available in the public domain.

One of the important challenges of creating the simulator for a DAG based DLT is the lack of resources to build upon. There currently exist no public simulators which can perform large scale simulation. There are however some very useful resources we found during the literature survey to base CIDDS on. Those resources are presented in this section.

3.2 Blockchain Simulators

There are quite a few blockchain based simulators available in the public domain. Some of the significant ones we came across were:

3.2.1 Bitcoin-Simulator

This is the first ever blockchain based simulator, which was proposed in [27] considers only blocks and it is based on a discreet event driven simulator, NS-3 [28]. This does not considers transactions.

3.2.2 VIBES

VIBES or Visualizations of Interactive, Blockchain, Extended Simulations [7] is another blockchain based simulator from Technical University Munich (TUM). This simulates thousands of nodes and proposes the concept of fast forwarding which is essentially the ability to skip ahead in time so that large scale simulations could be done in short time. This was built using AKKA's actor model [29].

This thesis was first proposed to be an extension of VIBES, but after thorough research, it is decided to make this a totally independent framework.

3.3 Tangle Simulation by IOTA

One of the most useful resources and the only publicly available DAG based simulators is IOTS's UI based simulator [30], which allows the users to generate Tangles with specified network parameters. A screencap of this simulator is presented in fig. 3.1.

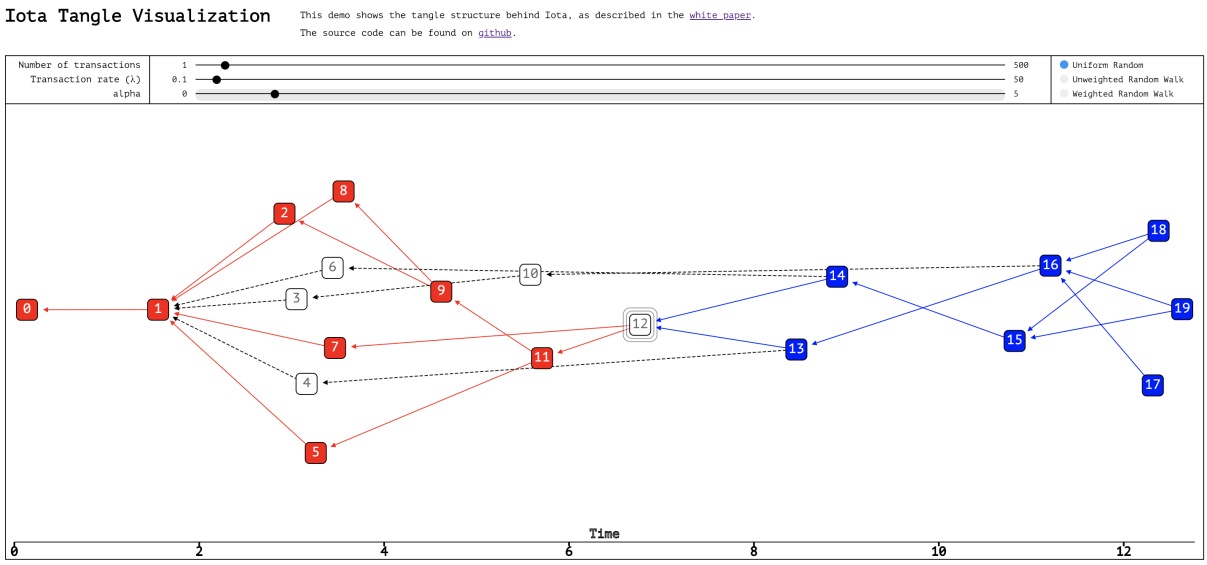


Figure 3.1: IOTA Tangle Simulation

This is purely a UI based simulator which lets the users to create and interact with smaller tangles (less than 500 transactions) and verify the properties. This was implemented using Node.js.

As shown in fig. 3.1 users can interactively change the network parameters and check the changes live.

One of the major difference between CIDDS and this simulation is the scale. CIDDS can simulate thousands of transactions.

3.4 Simulation by github user minh-nghia

During literature survey, we came across a github repository of a user minh-nghia where he simulates the IOTA's core algorithms with an aim to replicate the results [31]. This was just an personal project from the user to try and implement the core tip selection algorithms used in IOTA's tangle.

This was a very interesting find as the algorithms implemented were as close to IOTA's core implementation, yet simple to comprehend. Even though the simulator is not ideally suited for very large scale simulations and there is no web framework around the codebase (the code was written in Python using Jupyter Notebook [32]), the algorithm implementations were very insightful.

Chapter 4

Approach

One of the major requirement for the simulator is the need for it to be as close to real world as possible. For this, the issuance of transactions from the nodes or the Internet of Things (IOT) devices need to be independent of each other, and even thought the simulator uses a shared memory, the nodes should not be bothered about the use of shared memory and they should be behave like a truly distributed system.

Initially, this simulation framework was planned to be an extension of VIBES [7] framework, which allows simulation of large scale blockchain like systems. But after thorough review and a feasibility study, since a DAG based system does not closely follow the architectural principles defined in a blockchain based system, it was decided to be created as a separate simulation framework. Also, since researchers need to study the behavior and would like to compare the simulation results, it makes sense to have a web application as a wrapper for the simulation system, which could serve the simulation service to the users and allow storage and comparison in the future.

To gain some insights on simulating the Tangle, I joined the IOTA Discord channel [33]. Discord is a collaboration platform usually used by online gamers and IOTA has a channel where the core developers actively interact with those interested in IOTA. I got in touch with Alon Gal [34] from IOTA foundation and he provided a lot of valuable insights in developing this simulator. We would briefly discuss the pre-requites for the development in the following section most of which is inspired by the Tangle simulation by IOTA outlined in section 3.3, before getting into the development details.

As the outcome of all the discussions and research outlined above, we propose CIDDS, a configurable and interactive framework for simulating large scale directed acyclic graph based distributed ledgers.

4.1 Prerequisites

4.1.1 Poisson Point Processes

To create a close to real world system, the transactions issued from the participating nodes should arrive at random. One of the most commonly used mathematical model to define these random processes is Poisson Point Process. ”In probability, statistics and related fields, a Poisson point process is a type of random mathematical object that consists of points randomly located on a mathematical space” [35]. This is used to model random events, such as the customers who arrive at a store, phone calls to a service center, specific occurrence of events such as Earthquakes, etc distributed over a time space.

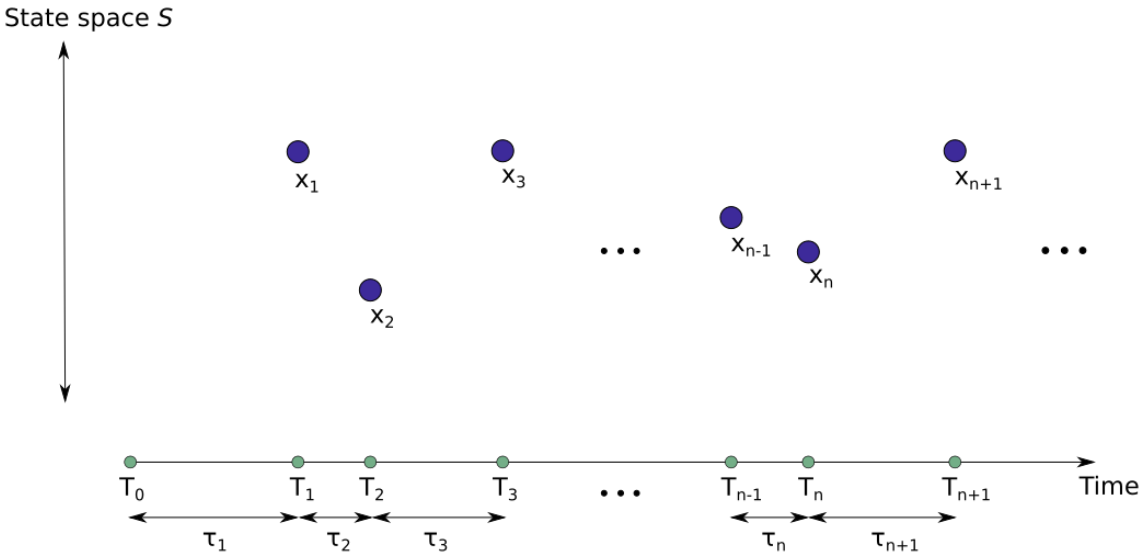


Figure 4.1: A typical Poisson Point Process

fig. 4.1 shows a typical marked Poisson Process [36], which we would be using to model the transactions in the simulator. Here, the points Xi take a value in the state space S, which is also called as a Mark Space. The unmarked point process is represented in the positive real line, in this case Time. This is a model to represent some random process Xi which arrives on a given Time Ti. With respect to CIDDS simulation framework, this Xi is always a transaction issued by a node and Ti is the time unit at which the transaction arrives at the system.

The average rate of incoming transactions in a Poisson Point Process is a constant, denoted by λ. This is a configurable parameter in CIDDS simulator. A small note is if λ is very low, the

DAG formed would be chain, because the number of transactions arriving is very low. A very large λ forms a cluster. This would be further discussed with examples in the thesis evaluation chapter.

4.1.2 Tip Selection Strategy

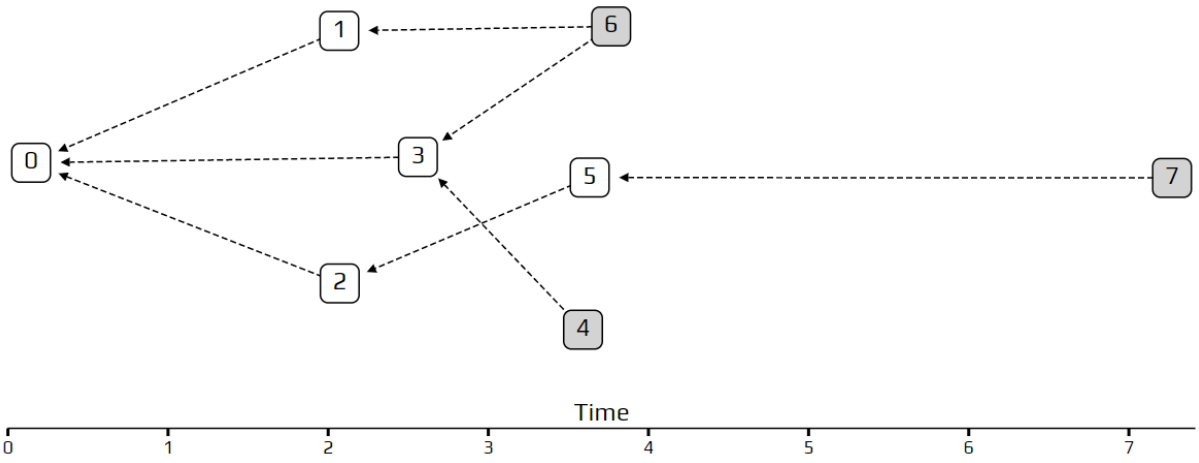


Figure 4.2: Tip Selection Illustration

All new transactions arriving in the system need to select a tip in the DAG, or a transaction to approve. This is vital because, this forms the basis of how the DAG grows and an optimal tip selection strategy is vital for prevention of too many orphans (unapproved tips) in a system. fig. 4.2 shows a typical tip selection process, where the arriving transaction 7, selects the visible tip 5 to approve. Hence it becomes the part of the DAG, with a directed edge from 7 to 5. CIDDS provides two distinct tip selection strategies: URTS and MCMC.

4.1.3 Random Walk

A random walk is a strategy employed to perform tip selection in the DAG. To achieve this, an imaginary ”walker” is placed on the genesis and moves towards the available tips by tracing a path starting from the genesis. The random walk can be weighted or unweighted. In unweighted random walk, from each node, the walker can go to any possible node with equal probability. In weighted random walks, the tips are selected based of a specific metric.

4.1.4 Conflict Resolution

Since the system uses a shared memory for the creation of the DAG, there may be conflicts during run time for access to same resources (e.g write to the DAG). Due to this, we need a conflict resolution mechanism. This is achieved using a component called as Process Orchestrator, which handles scheduling of tasks and conflict resolution.

4.1.5 Configurable Parameters

Since the users studying the behaviour of the DAG need to isolate specific properties and study the behaviour of the system, the system needs to be highly configurable. A list of parameters which the users can tweak is presented in table 4.1. The user can change any of these parameters in isolation or combine multiple parameters to check the effect of the change.

Configurable Parameters	**Symbol**	**Description**
Number of Processes	-	Total number of devices to be simulated
Number of Transactions	n	Total number of transactions in the DAG
Transaction Rate	λ	The average number of incoming transactions per unit time
Level of Randomness	α	Parameter to force a uniform tip selection
Tip Selection Algorithm	-	The algorithm for tip selection strategy

Table 4.1: CIDDS - Configurable Parameters

4.2 Design of Proposed Solution

To achieve the desired outcome, the design follows an approach were we create a simple directed acyclic graph, which is shared in memory by various threads, each analogous to the IOT device which issues the transaction. To achieve this, mostly all the complex calculations required to issue the transaction like PoW operations discussed in [1] are skipped. This helps in focus on the other network parameters defined in table 4.1.

This Directed acyclic graph is represented by a Class DAG. This contains classes Transaction and Genesis, which is a special transaction, issued as the very first transaction. In addition, all

the configurable parameters are defined in the class DAG as standard attributes. These attributes can be extended to include further parameters as discussed in section 6.3.

4.2.1 Architecture

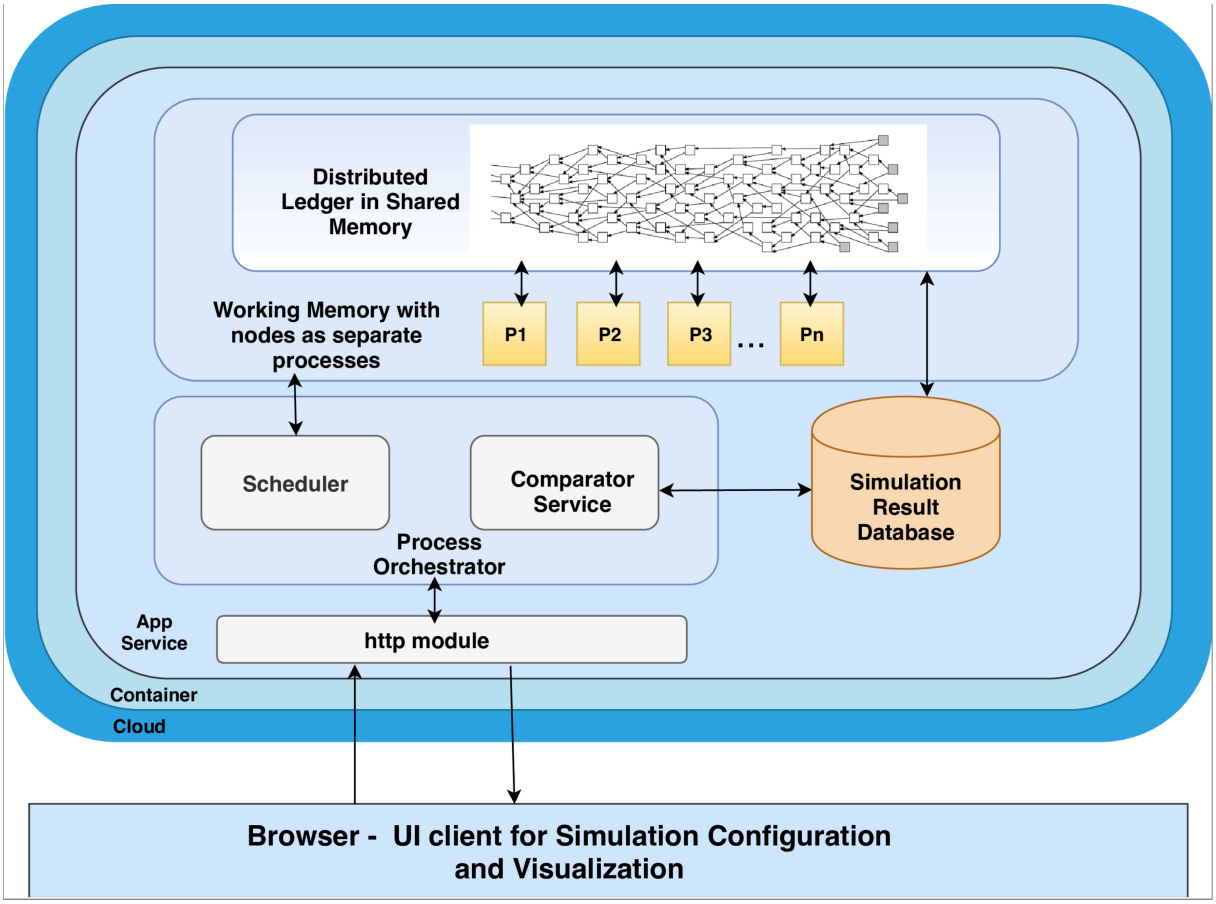


Figure 4.3: CIDDS - Architecture

The architecture of CIDDS is shown in fig. 4.3. Since the simulation framework was decided to be developed as a web application, it needs to have two distinct components namely:

- **Simulator Base** :
  This is the core component which encompasses the algorithms and processing logic needed to run the simulator. The core is wrapped by an application layer.
- **Application Layer** :
  The Application layer runs on top of the base. This is the application which is rendered in the Browser and allows the user for interaction with the user.

Further to disuss fig. 4.3 the individual components of which make up CIDDS are detailed below:

Browser Client

The browser client renders the UI for the user interaction. By design, the application should run on all modern browser. Also, the design is asynchronous so that the client does not wait for the backend to finish processing the simulation as this might take a lot of time. The user would be able to perform the following actions on the browser client.

1. **Login/Logout**: The browser would support user authentication and user management in general.
2. **Initiate a simulation**: The ui client would allow the user to initiate the simulations by specifying the parameters.
3. **Check Status**: The ui would also allow the user to check the status of the running simulations. Also the user can refer to an old simulation using the simulation history.
4. **Compare Simulation**: The ui client would also assit the user in comparing two simulations side by side

   The implementation details of the browser client is discussed in detail on subsection 4.3.3.

HTTP Module

The frontend needs to communicate with the backend using a http connection. For this a http server needs to run at the backend. This server forwards the requests to the simulator. Standard Hypertext Transfer Protocol over Secure Socket Layer (HTTPS) is used for server calls. The routing and serving pages is handled by the Django server.

Container and App service

The simulator is wrapped in an app container and served via web server. This also handled by Django. And we run a NGNIX [37] application platform to serve the container. The container is configured with standard configurations for serving a web application and there are no special additions for CIDDS itself. The app service is provided by Django server.

Process Orchestrator

Process orchestrator forms the core of CIDDS simulator. This acts like a co-ordinator for all the requests to the Django server. Since multiple simulations can be run at the same time, it is vital for the simulator to make sure resources are used properly. The orchestrator does this by scheduling the simulations and performing conflict resolution when necessary. The process orchestrator contains two important subcomponents, as shown in fig. 4.3.

1. **Scheduler**: The scheduler service is needed for scheduling the simulation requests. The scheduler module prioritizes the requests, puts them into a scheduling queue when the simulator's resources are under use. The requests are honored in First Come First Serve (FIFO) manner.
2. **Comparator Service**: Once the simulations are done, they are stored in the databases (more about this in subsection 4.2.1). The process orchestrator facilitates this by implementing a comparator service, which enables two simulations to be compared. This can be enhanced to support comparison of multiple simulations, this is discussed in section 6.3.

Working Memory

In CIDDS, working memory refers to the simulation memory, where the process threads are spawned to simulate the IOT devices issuing the transactions. These individual threads act independent of each other, and generate a DAG. This DAG closely follows most of the principles outlined in [1]. The distributed ledger grows in the memory and the working memory and abides by the consensus protocols defined. Once a simulation is complete, this DAG is persisted in a database and the working memory is cleared.

Simulation Result Database

A database is vital for the simulator as real time interaction with large scale simulations is impossible as the simulations are huge and might take hours to complete. Hence all simulations are stored in a database and then used later for comparison. CIDDS uses postgresSQL [38] as its primary database as this is a versatile, yet simple to use database.

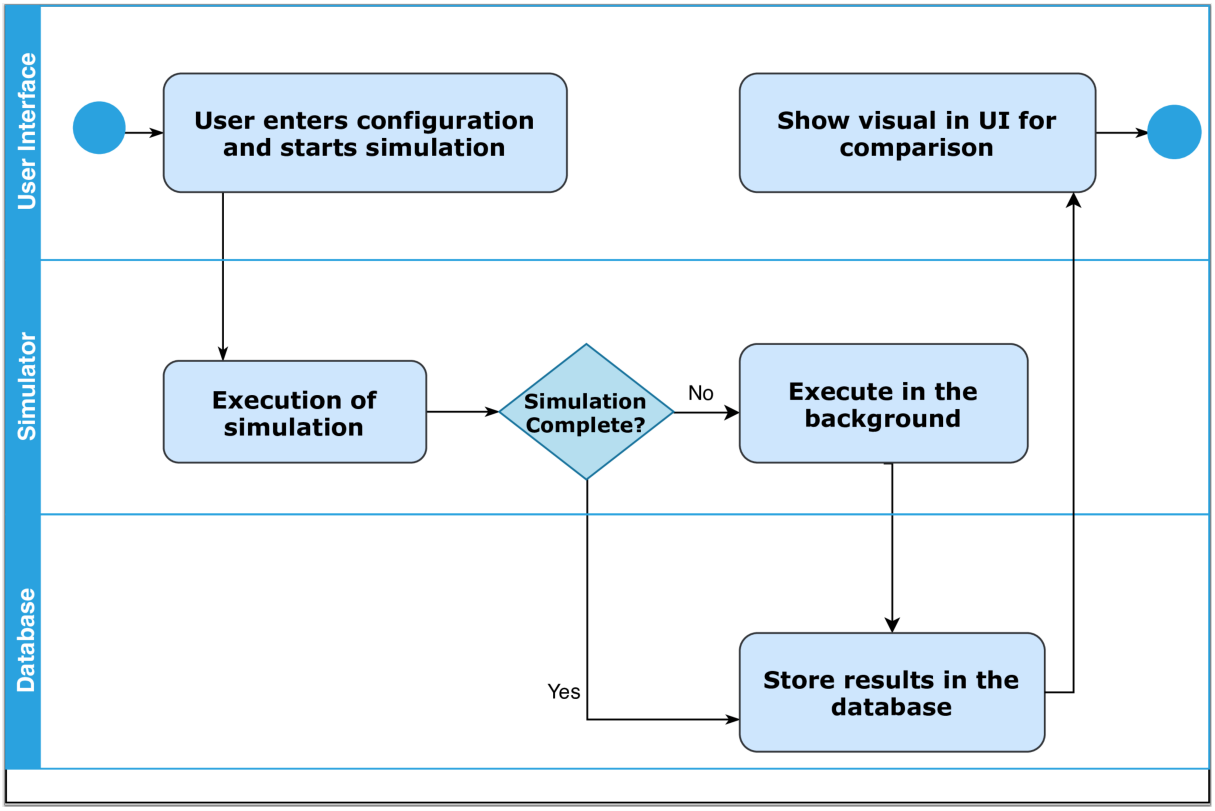


Figure 4.4: CIDDS - Simulation workflow

4.2.2 Simulation Workflow

The chart represented in fig. 4.4 shows the workflow of a single simulation in CIDDS. The flow if the actions from initiation to completion is represented in the following steps.

Step 1: The user enters the configuration parameters and initiates the simulation. Usually the simulation is accompanied with a simulation reference for later reference.

Step 2: The Process orchestrator of the simulator receives this request and starts the execution of the simulation.

Step 3: After a predefined wait time, the orchestrator checks if the simulaition is complete.

Step 4: If the simulation is complete, the simulation status is marked as ”Done” and stored in the database.

Step 5: A visual representation of the simulation is generated, usually an image and shown to the user via the user interface.

Step 6: If the simulation still runs, which is the case for the large scale simulations, the processing is pushed to the background and execution continues.

Step 7: Once this simulation is complete, steps 3 and 4 follow.

Status	**Description**
Initiated	A simulation which has just started by the scheduler
Running	A simulation which is being executed in the background
Complete	A simulation which is executed and saved in DB
Failed	A simulation failed due to errors

Table 4.2: CIDDS - Possible Statuses

A list of all the possible statuses which a simulation can have is represented in table 4.2. Also, not represented in the fig. 4.4 is the case where the user wishes to check the status of the simulations or compare the simulations. This can be done at any time the user wishes and is not part of the core work flow.

4.3 Implementation

In this section, we discuss the implementation details of CIDDS. The implementation closely follows the design explained previously. For implementation, we followed an Iterative model [39] of Software Development Life Cycle (SDLC). Continuous feedback from and discussions with the advisor is used to adjust the requirements based on the literature survey and research. One of the most important challenges faced during the implementation is the lack of resources to refer, because DAG based DLT is relatively new. Most of the inputs for developing this simulator came from discussions in the iota discord channel [33].

4.3.1 Technology Choice

For creating a simulator of DAG based DLT, we do not need the sophisticated block creation mechanism and network propagation needed for a blockchain based simulator. Initially, CIDDS was planned to be an extension of VIBES [7]. VIBES is a simulator for blockchain based systems and was written based in Akka based Actor model [29]. This is a perfect choice

for blockchain based systems and provides a high parallelization of execution. But this also introduces additional overhead which is unnecessary for a simulator like CIDDS.

We chose Python as a programming language of choice for CIDDS because of the following properties inherent to the language and platform:

- **Easy syntax and readability**:
  Python language is one of the most popular language with a syntax resembling pseudo code and the code is highly readable, which would be an advantage for a open source project like CIDDS.
- **Object Oriented High Level Language**:
  Most of the design elements of CIDDS demand a object oriented approach as would be discussed in the . Also, python being a high level language supports rapid prototyping and hence multiple solutions can be tried and tested within the short timeframe the thesis is undertaken.
- **Free and Cross platform support**:
  CIDDS is hosted in a Ubuntu server and it was developed using Mac OS. This cross platform support is vital for an open source project. Also support for most functionalities CIDDS use like graphs, network generation, etc are supported via libraries.
- **Extensiblity**:
  Python also contains a lot of free add-ons, libraries and toolkits. This would assist a lot in future extensibility of the project if CIDDS needs to be accommodating simulations of other DAG based DLTs like Hashgraph [10].

Thus Python was a clear choice to work with. Once this is decided, we needed to find the frameworks to employ to achieve the objectives of CIDDS. There were two distinct design choices. The frameworks needed for developing the core of the simulator which comprises of process orchestrator and the tip selection algorithms.

Python for Core Algorithms

The core algorithms are developed using standard Python. Two of the important libraries used in the development are networkx [40] and Python Pathos [41].

- **Networkx**: ”NetworkX is a Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks” [40]. This is highly helpful for CIDDS because DAG is supported natively by Networkx.

- **Python Pathos**: ”Pathos is a framework for heterogenous computing. It provides a consistent high-level interface for configuring and launching parallel computations across heterogenous resources. Pathos provides configurable launchers for parallel and distributed computing, where each launcher contains the syntactic logic to configure and launch jobs in an execution environment” [41]. Since in CIDDS we need parallel computing to simulate the multiple IOT devices, pathos is a good choice.

Once the framework for core algorithms is finalized, we needed a wrapper around this core to present the simulation information to the user.

Django Wrapper

Django [11] is one of the most popular web development framework, which allows rapid protoyping of ideas. In addition to prototyping, Django can also be used to create production ready application as many popular websites like Youtube, Instagram, Mozilla, Pintrest [42] use Django. Other major advantages [43] of using Django are:

- **Speed**: Django is probably one of the fastest frameworks to get an application from protype to production
- **Built-in Packages**: Django comes with nifty packages such as auth, admin, templates, etc. These make the life of the developer easy
- **Security**: Django is secure against almost all of the common security issues like Cross Site Request Forgery (CSRF), Cross site scripting, SQL injection, clickjacking etc.
- **Scalability and versatility**: Django scales well. Scalability is an important aspect of CIDDS as it needs to handle large scale simulations.

Also, I already had some experience with using Django framework and hence it was a clear choice for me to use for the developemnt of CIDDS.

The following sections detail the entire development process.

4.3.2 Backend

With django, a project with the directory structure shown in fig. 4.5 is created. The general directory structure closely resembles a standard Django project [44]. We would not discuss the django directory structure here, instead we would focus on the most important aspects of CIDDS.

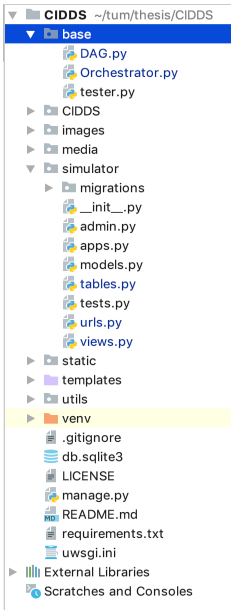


Figure 4.5: CIDDS - Project Structure

CIDDS base

The base forms the core of CIDDS detailed in fig. 4.3. This contains the DAG class, the process orchestrator and a tester module for running the tests on the application. Lets detail each component in the base:

1. **DAG Class**: The DAG file contains the core DAG with the attributes defined in table 4.3. The core encapsulates the DAG which needs to be generated in the shared memory with all its associated functions. The core algorithms are implemented here.

   The DAG is initialized using a special transaction named as Genesis. Since genesis is the first ever transaction, it has special properties like no need to approve any nodes. Also every tip would directly or indirectly approve the genesis transaction in a DAG.

   The DAG also contains a Transaction Class, which denote the transactions arriving in the system other than the genesis transaction.

2. **Process Orchestrator**: The process orchestrator performs the routing of requests to the core and initiates the simulation using a helper method. The code snippet for this method is presented in listing 4.1. The starthelper initiates the plot and then executes the simulation. Also, the time traces are added in the orchestrator to have a record of execution time of the simulation.

Parameter	Description
genesis	Holds the genesis block of the DAG
transactions	Contains the transactions of the DAG
time	Refers to the time units taken for the simulation
rate	Rate of incoming transactions
alpha	Degree of Randomness
algorithm	The algorithm for tip selection strategy
cache	Temporary storage of transactions for exection
nodes	List of all transactions for UI rendering
links	Edges between the transactions for UI rendering
stepcounter	An Incremental Counter

Table 4.3: CIDDS - DAG Fields

```
def start_helper(sim):

    '''

    :param sim: simulator object
    :return: dag : DAG
    '''

    # Set plots for resulting dag image
    plt.rc('axes', labelsize=20)
    plt.rc('xtick', labelsize=20)
    plt.rc('ytick', labelsize=20)

    plt.figure(figsize=(25, 10))

    # To Trace execution time
    start_time = time.time()
```

```
    # Call the DAG to generate transactions
    dag = DAG(rate=sim.alpha, algorithm=sim.algorithm, plot=True)
    for i in range(sim.transactions):
        dag.generate_next_node()

    print(" Execution Time : %s seconds ---" % (time.time() - start_time))

    # Return the result
    return dag
```

Listing 4.1: Process Orchestrator - Start helper

The conflict resolution mechanism is outlined in the resolveConflict method provided in the snippet listing 4.2. This is achieved by having a locking mechanism and a short queue

```
def _p_resolveConflict(self, sim, t):
    '''

    :param self: Orchestrator
    :param sim: simulation object
    :param t: executed thread
    :return: simulator
    '''

    # Lock the current thread
    lock = threading.Lock()
    lock.acquire()
    try:
        # Initiate simualtion
        self.start_helper(sim)
    except Exception as ex:
        # Handle error
        thread_name = threading.current_thread().name
        pprint("Error in initiating simulation {}.{}".format(thread_name, ex))
    finally:
        # Release lock after execution
        lock.release()
```

```
        return sim
```

Listing 4.2: Process Orchestrator - Conflict Resolution

Django HTTP Handler

For separating the core logic from the request handling, a new app is created in django, named as ”simulator” as shown in fig. 4.5. To handle all the http requests, django routes the requests to views.py of ”simulator” app. Views.py contains handlers for GET and POST requests. The code snippet is available in subsection A.2.4.

4.3.3 Frontend

The front end is created using django templates [45]. Being a web framework Django supports creation of beautiful UI out of the box using Django Templating Language (DTL). This is done by configuring the django templates in settings.py of CIDDS application as shown in listing 4.3. The source code of entire settings module is presented in the appendix.

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates')]
        ,
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

Listing 4.3: Django Templates - Settings

In these settings there are two important aspects to note:

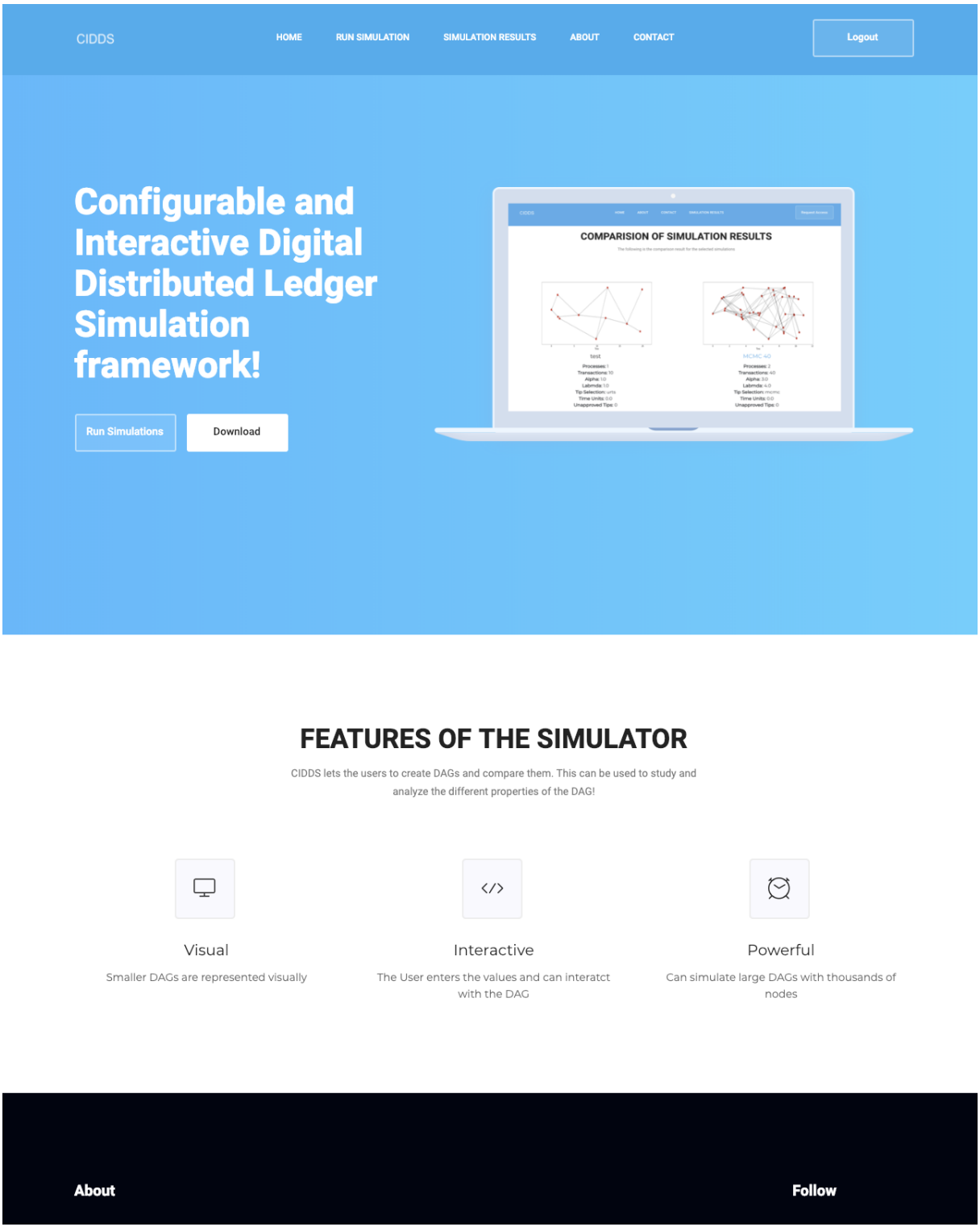


Figure 4.6: CIDDS - Homepage

- **DIRS**: This is the list of directories where django engine looks for template source files, in the case of CIDDS, it is in the directory *templates*.
- **APP_DIRS**: This is True because the app engine should look for templates in all the applications
- **OPTIONS**: This contains backend specific options.

Once these options are configured, it is fairly straightforward to obtain beautiful visuals combined with backend specific options.

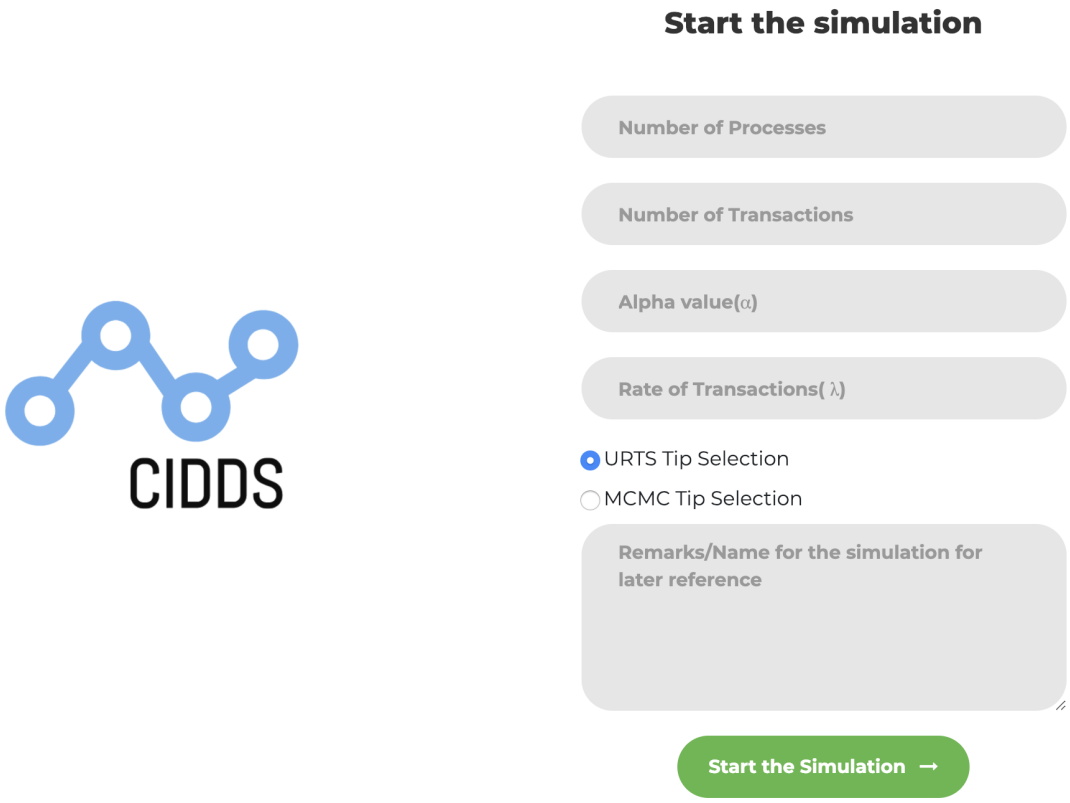
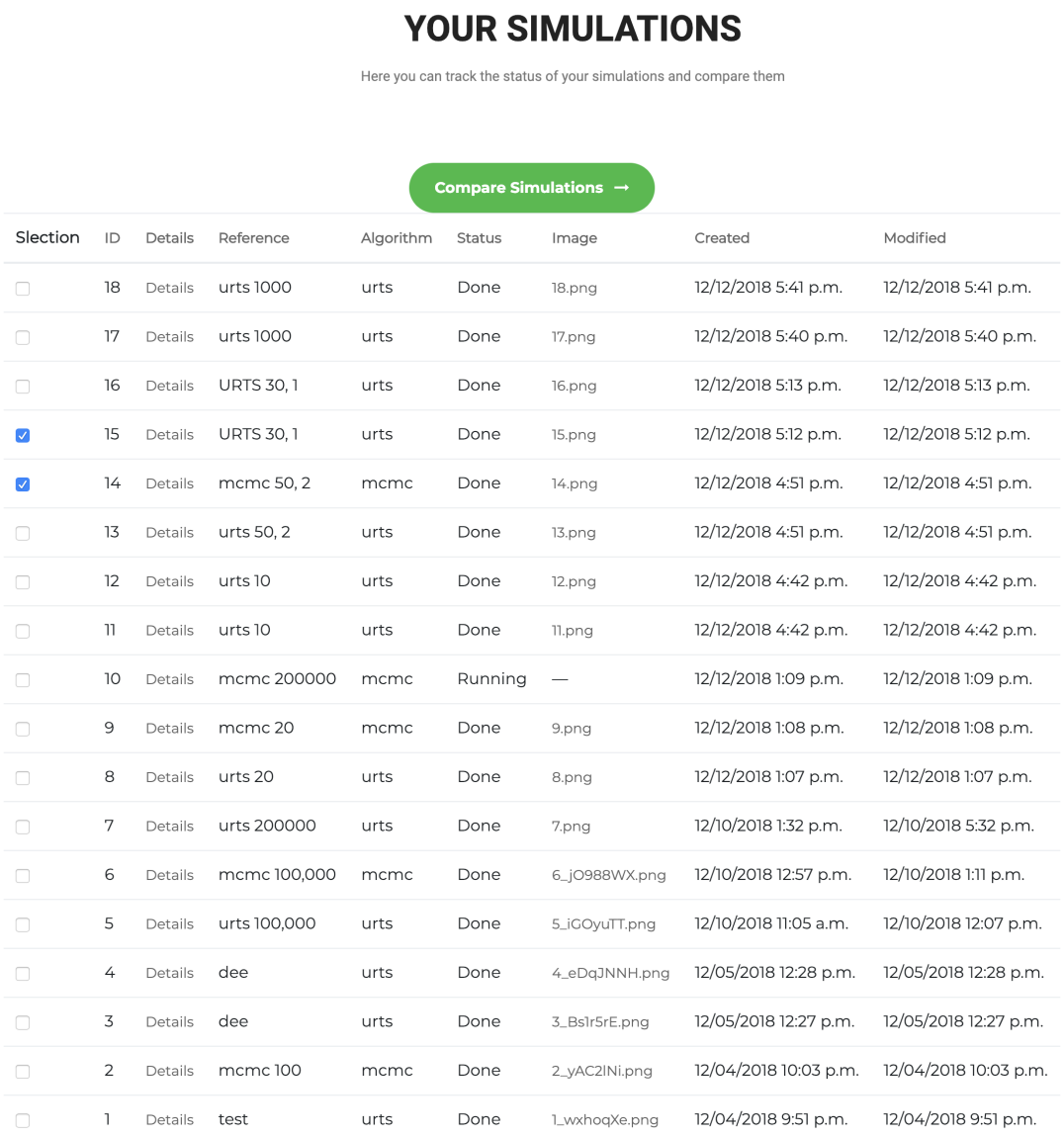


Figure 4.7: CIDDS - Simulation Startform

Some of the most important visuals in CIDDS are:

Home Page

The homepage, a snapshot of which is presented in fig. 4.6, is the entry point into the simulator. This webpage is made responsive and intuitive. It lists all the options which are available to the user within the simulator.



Slection	ID	Details	Reference	Algorithm	Status	Image	Created	Modified
☐	18	Details	urts 1000	urts	Done	18.png	12/12/2018 5:41 p.m.	12/12/2018 5:41 p.m.
☐	17	Details	urts 1000	urts	Done	17.png	12/12/2018 5:40 p.m.	12/12/2018 5:40 p.m.
☐	16	Details	URTS 30, 1	urts	Done	16.png	12/12/2018 5:13 p.m.	12/12/2018 5:13 p.m.
☑	15	Details	URTS 30, 1	urts	Done	15.png	12/12/2018 5:12 p.m.	12/12/2018 5:12 p.m.
☑	14	Details	mcmc 50, 2	mcmc	Done	14.png	12/12/2018 4:51 p.m.	12/12/2018 4:51 p.m.
☐	13	Details	urts 50, 2	urts	Done	13.png	12/12/2018 4:51 p.m.	12/12/2018 4:51 p.m.
☐	12	Details	urts 10	urts	Done	12.png	12/12/2018 4:42 p.m.	12/12/2018 4:42 p.m.
☐	11	Details	urts 10	urts	Done	11.png	12/12/2018 4:42 p.m.	12/12/2018 4:42 p.m.
☐	10	Details	mcmc 200000	mcmc	Running	—	12/12/2018 1:09 p.m.	12/12/2018 1:09 p.m.
☐	9	Details	mcmc 20	mcmc	Done	9.png	12/12/2018 1:08 p.m.	12/12/2018 1:08 p.m.
☐	8	Details	urts 20	urts	Done	8.png	12/12/2018 1:07 p.m.	12/12/2018 1:07 p.m.
☐	7	Details	urts 200000	urts	Done	7.png	12/10/2018 1:32 p.m.	12/10/2018 5:32 p.m.
☐	6	Details	mcmc 100,000	mcmc	Done	6_jO988WX.png	12/10/2018 12:57 p.m.	12/10/2018 1:11 p.m.
☐	5	Details	urts 100,000	urts	Done	5_iGOyuTT.png	12/10/2018 11:05 a.m.	12/10/2018 12:07 p.m.
☐	4	Details	dee	urts	Done	4_eDqJNNH.png	12/05/2018 12:28 p.m.	12/05/2018 12:28 p.m.
☐	3	Details	dee	urts	Done	3_Bs1r5rE.png	12/05/2018 12:27 p.m.	12/05/2018 12:27 p.m.
☐	2	Details	mcmc 100	mcmc	Done	2_yAC2lNi.png	12/04/2018 10:03 p.m.	12/04/2018 10:03 p.m.
☐	1	Details	test	urts	Done	1_wxhoqXe.png	12/04/2018 9:51 p.m.	12/04/2018 9:51 p.m.

Figure 4.8: CIDDS - Simulation History Visual

Only the authenticated user has the option to run simulations. This is achieved by clicking on ”Run Simulations” button and this takes the user to the second page, which is start form.

Simulation Start form

The simulation start form, which is displayed in fig. 4.7, allows the user to enter the simulation parameters and initiate the simulation. This is fairly straight forward and simple to use form with validations.

Simulation History

Once a simulation is initiated the user is redirected to the history page which is depicted in fig. 4.8. Here the user can obtain the running status of the simulation and have the option to look into the details. Also the users can compare the simulations by selecting two simulations and clicking on ”Compare Simulations” button. This takes the user to the comparison page, which is depicted in fig. 5.7.

The users also have the option to check the visual representation of the simulation in case of smaller simulations. Also, a intuition of the time taken for simulation can be obtained by looking into the simulation start time and last changed time fields.

Again, the user could potentially extend the views to add their own visuals and more of this is discussed in section 5.4.

4.3.4 Deployment

Ngnix Webserver Configuration

Once the simulator is ready, it had to be deployed in server for public access. This was achieved by running the django in an ubuntu server available with Leibniz-Rechenzentrum (LRZ). Once the server was setup and running in background, Ngnix [37] webserver was installed in the ubuntu server and configured with the settings in listing 4.4.

```
server {
    listen 80;
    server_name cidds.blockchainsimulator.org
    www.cidds.blockchainsimulator.com;

    location = /favicon.ico { access_log off; log_not_found off; }
    location /static/ {
        root ***HIDDEN****;
    }

    location / {
        include uwsgi_params;
        uwsgi_pass unix:***HIDDEN***/cidds/cidds.sock;
    }
```

```
}
```

Listing 4.4: Ngnix Configuration

Database Configuration

In addition to installing and configuring django and ngnix, for running CIDDS, posgreSQL [38] had to installed and the database had to be setup. This is achieved by configuring PostgreSQL in the settings.py of CIDDS app as shown in listing listing 4.5.

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': "cidds",
        'USER': "postgres",
        'PASSWORD': ***HIDDEN***,
        'PORT': '5432',
        'HOST': 'localhost',
    }
}
```

Listing 4.5: PostgreSQL - Settings

PostgreSQL runs on the default port 5432 and the database is named as ”cidds”. All the database tables are autogenerated using migrations available within Django.

Chapter 5

Evaluation

For evaluation we are going to use the parameters discussed in chapter 1. The experiments are conducted in two different machines:

1. **Machine 1**: CIDDS running on a Macbook Pro with configurations - 2,3 GHz Intel Core i5 and memory - 8 GB 2133 MHz LPDDR3.
2. **Machine 2**: CIDDS deployed and running on an Ubuntu server with a total virtual memory of 34359738367 kB and 4 processors each with specifications Intel(R) Xeon(R) CPU E5-2697A v4 @ 2.60GHz.

Also please note that the parameter α has no effect on the simulation if the tip selection algorithm used is URTS as the tip selection is uniform in URTS.

Following are the methodology and results of the conducted evaluations.

5.1 Correctness

The CIDDS simulator is designed to simulate the DAG as closely as defined in the Tangle whitepaper [1]. The simulation results are closely analogous to the Tangle discrete simulation results published by IOTA foundation [9].

In the given DAG, every transaction needs to directly approve two transactions, except the genesis transaction. Let's assume that transaction A directly approves a transaction B and it is denoted as A $\rightarrow$ B. Then A indirectly approves a transaction Z if there is atleast a sequence of 3 transactions which satisfy the condition:

$$A \rightarrow B \rightarrow ... Z$$

In the simulation, we assume that all the transactions have a constant weight. Also, the unit of time used in the simulation is a time step. When the simulation is run, there is an average of λ transactions issued. The number is always chosen from the Poisson Distribution. Every transaction generated and introduced into the system always approves atmost two older transactions directly, i.e any transaction issued in time step t, always approves a transaction issued between 0 and t-1 time step. Also, all the calculations needed to issue the transaction are skipped and are assumed to be done within a single timestep. Four sample simulations with 10 or 100 transactions (smaller number of transactions for clarity) are done with the two different tip selection algorithms and the results are presented here. In all the simulations the other parameters are kept constant at: processes = 1, α =1, λ =1. All these experiments are done in Machine 2.

5.1.1 Simulation 1 - URTS Tip Selection, n = 10

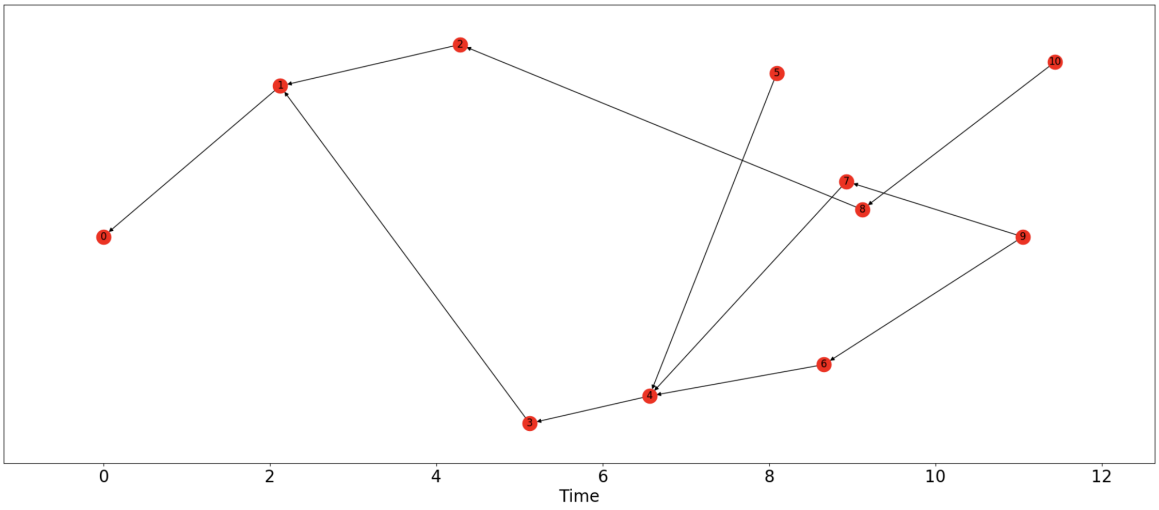


Figure 5.1: CIDDS Evaluation Correctness - Simulation 1

URTS algorithm selects the tips at random (with uniform distribution). In fig. 5.1 0 is the genesis and transaction 1 directly approves the genesis. Transaction 10 also approves the genesis as it satisfies the sequence of atleast 3 transactions from 10 to genesis. In this case (10 $\rightarrow$ 8 $\rightarrow$ 2 $\rightarrow$ 1 $\rightarrow$ 0).

Also, 8 is a lazy transaction as it approves a very early transaction 2 due to random nature of tip selection.

5.1.2 Simulation 2 - URTS Tip Selection, n = 100

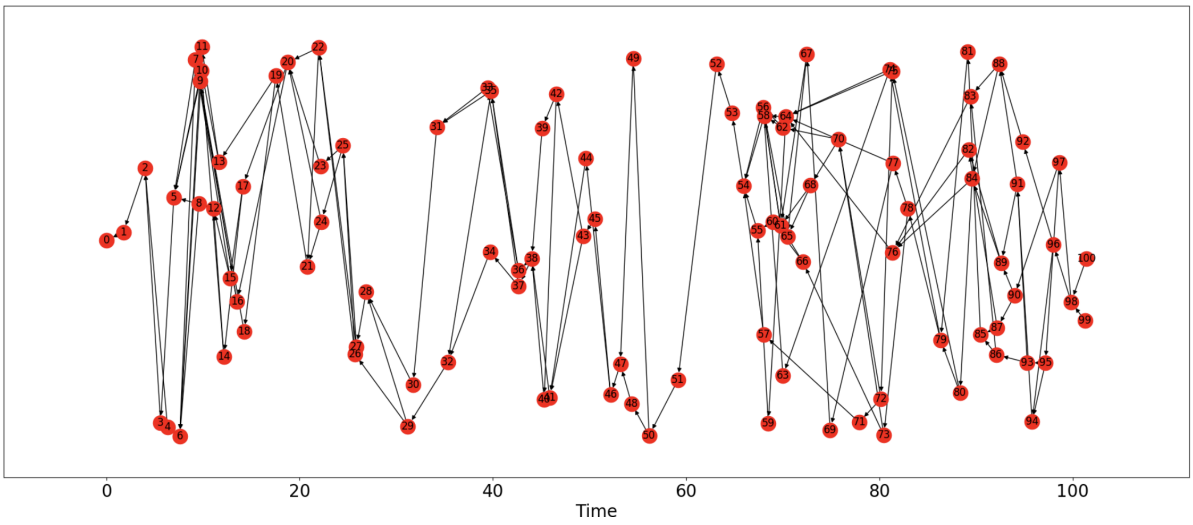


Figure 5.2: CIDDS Evaluation Correctness - Simulation 2

Simulation 2 also runs URTS, but with more transactions. Here it is clear that with the increasing transactions, approval confidence increases. In this case, lets say transaction 1 has a lot more transactions approving it (All 98 transactions arriving after 1) and hence it can be said fairy confident that 1 is a valid transaction in the tangle.

5.1.3 Simulation 3 - MCMC Tip Selection, n = 10

The MCMC algorithm uses a random walk of particles towards the tip. In case of 10 transactions, the difference between urts and mcmc is barely noticeable. But still we could clearly see that the transactions don't tend to be lazy and approve transactions which are fairly recent in the timeline. Lets take an example of the transaction 9 from fig. 5.3. The approval from 9 to genesis 0 can be noticed to be not lazy.

$$9 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 0$$

5.1.4 Simulation 4 - MCMC Tip Selection, n = 100

This simulation as depicted in fig. 5.4 shows MCMC algorithm where tips are selected based on weighted random walk. Again, lazy tips tend to be less with selecting MCMC.

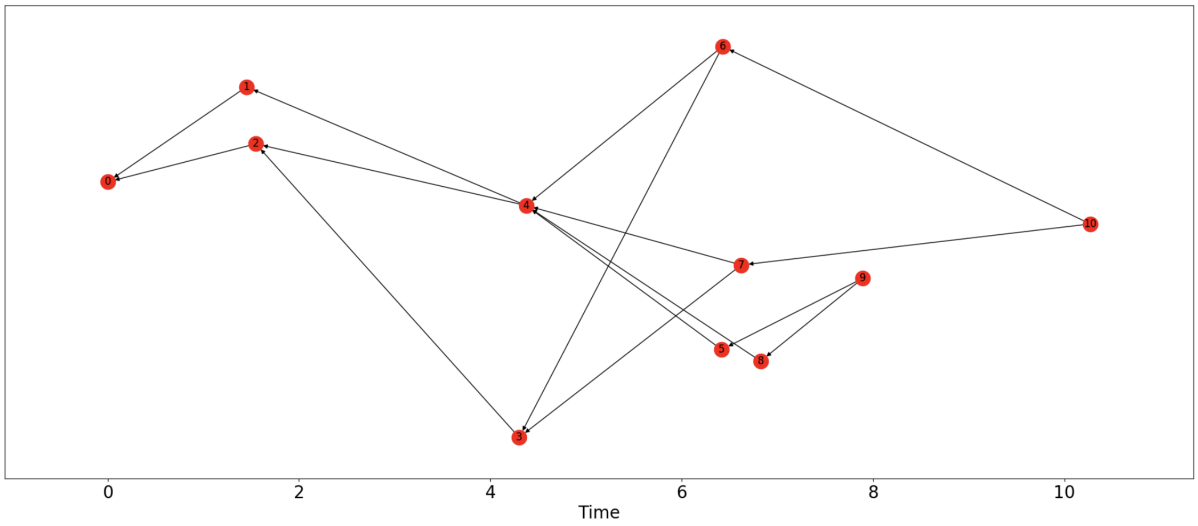


Figure 5.3: CIDDS Evaluation Correctness - Simulation 3

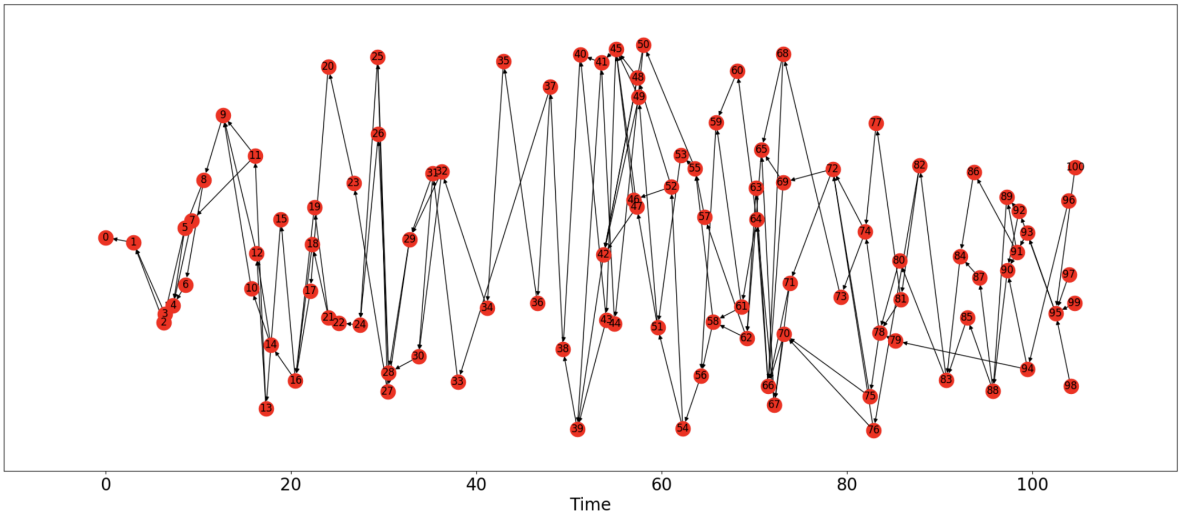


Figure 5.4: CIDDS Evaluation Correctness - Simulation 4

5.1.5 Results

The results of these 4 simulations are collated and presented in table 5.1. As an outcome of these simulations, it can be stated that the implementation is fairly closer to the DAG defined in the Tangle whitepaper [1].

Simulation	Transactions	Tip Selection	Time	Unapproved Tips
1	10	urts	11.43	4
2	100	urts	101.3	1
3	10	mcmc	10.2	3
4	100	mcmc	104.6	2

Table 5.1: CIDDS Evaluation - Correctness Simulation results

5.2 Speed

Speed of execution is another important parameter is evaluating simulator. Although the speed is largely dependent on the machine in which the simulation is executed, we expect a linear increase in time of execution with the increase in number of transactions. To demonstrate this, two sets of experiments are done on Machine 1 and Machine 2 mentioned above and the results are presented here.

The traces are done using Python's inbuilt time module. The start time is captured before the start of simulation and the end time captured after the end of simulation and the difference in seconds:

$$\text{execution time} = \text{start time} - \text{end time}$$

is used as a metric to evaluate the total execution time of a single simulation. The plots are created using an online plot generator[46].

5.2.1 Experiments on Machine 1

For this experiment, we vary only the number of transactions keeping every other parameters constant (processes = 1, α =1, λ =1, tip selection = mcmc). The visual representation of the results are presented in graph. Tabular form of the results with individual tranasction vs time value is available in the appendix. As can be seen from the fig. 5.5, the execution time of CIDDS increases with the transactions linearly. Also, another important aspect to note is that executing a simulation of a DAG with 300k nodes take a little over 6 hours due to the hardware limitations of the machine.

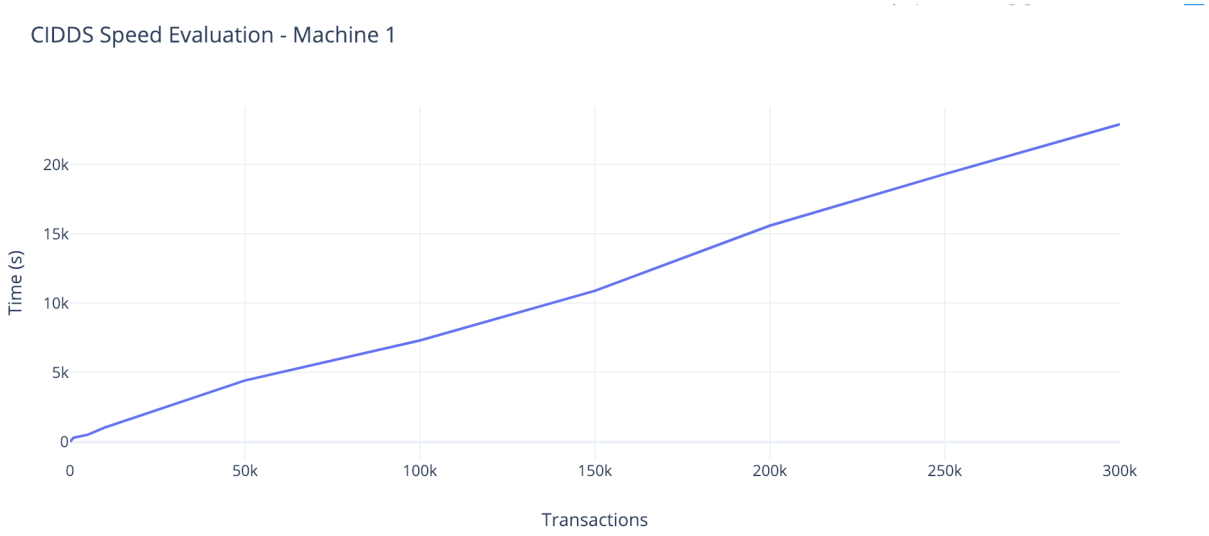


Figure 5.5: CIDDS Evaluation Speed - Machine 1

5.2.2 Experiments on Machine 2

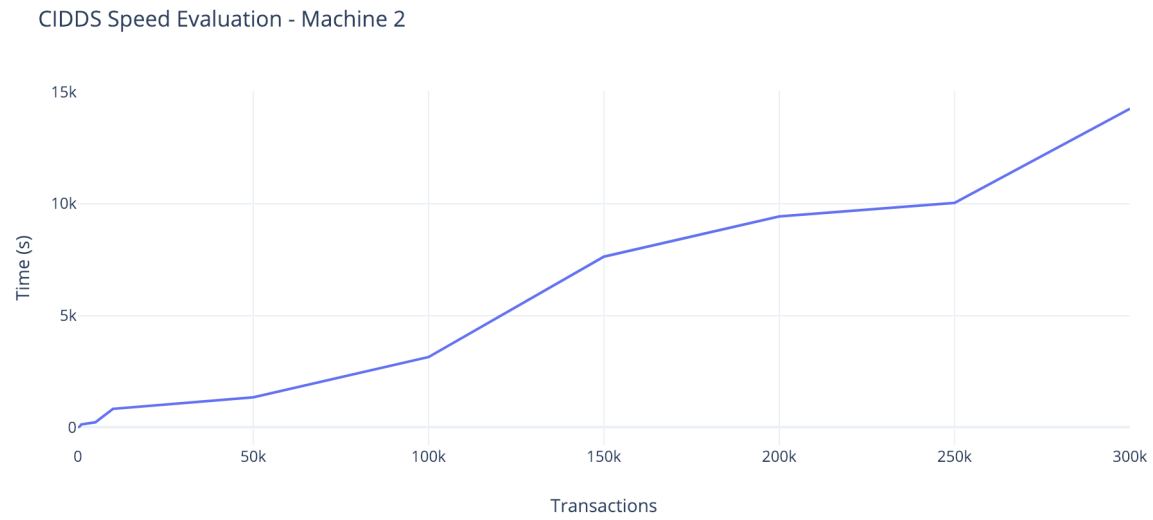


Figure 5.6: CIDDS Evaluation Speed - Machine 2

This experiment follows the same setup as subsection 5.2.1, but on a deployed Ubuntu server (Machine 2). The results are represented in fig. 5.6 and this can also be seen as fairly linear. We notice some variations on the time of execution and this may be attributed to the presence of multiple processor cores in the target machine. Overall, we get a liner increase in execution time with increase in transactions.

Also, the execution times are faster as expected because of the underlying hardware. Here one

aspect to note is that a execution of 300k transactions takes a little under 4 hours. Theoretically, this can be further improved by utilizing the multiple cores of the machine and this is discussed in section 6.3.

5.2.3 Result

As expected the execution speed is linear and we don't see any erratic behaviour in the simulator. Also, CIDDS is probably the only publicly available simulator at the time of writing the thesis which can handle thousands of nodes in a DAG.

5.3 Scalability

We expect a linear scalability with increasing number of nodes and the experiments conducted in section 5.2 clearly indicate that with increse in n (number of transactions), CIDDS scales with a time complexity of O(n). With respect to other parameters such as α and λ, the simulator scales linearly with a time complexity of O(1).

5.4 Extensibility

Since CIDDS is intended as a simulator to be used by researchers who would like to study specific aspects of the DAG, extensibility is vital. We would like to demonstrate the extensibility of CIDDS via 2 case studies.

5.4.1 Case Study 1 - Adding an additional attribute to DAG

If a researcher studying CIDDS wishes to add an additional attribute to the DAG, for example, transaction size, which is calculating the total physical memory consumed by a single transaction, and inturn calculate the total memory which a DAG would consume in run time, they need to perform the following changes:

Step 1: **Change DAG.py** : Lets assume the user would like to call the total memory consumed by a single transaction as transaction_size (represented in kB). They need to add this attribute in the Transaction class of the DAG.

Once this change is added, they can change the class DAG to include the attribute total_size.

Then we could add a method:

```
def calculate_totalsize(self, transaction_size):
    '''

    :param self: DAG
    :param transaction_size: Size of the arriving transaction
    :return self.total_size: Total size of the DAG
    '''
    self.total_size += transaction_size
    return self.total_size
```

Listing 5.1: CIDDS Extensibility - Case study 1

which calculates the total size of the DAG dynamically with every incoming transaction.

Step 2: **Change models.py of simulator app** : If the user wishes to persist this value in the simulator and see in the results, they just need to add an additional attribute to the models.py file of simulator app in the Django project.

```
# Total size of the DAG in KB
total_size = models.FloatField(blank=False, null=False, default=0)
```

Listing 5.2: CIDDS Extensibility - Case study 1

Step 3: **Changes in views.py of simulator app** : Once the above steps are complete, the user can use the field total_size in the needed GET or POST methods.

Thus a single field extensibility is fairly straight forward in CIDDS.

5.4.2 Case Study 2 - Adding an new functionality to DAG

In this case study, we explore the possibility of adding a totally new functionality to the DAG - for example, implementation of a PoW mechanism.

Since for the sake of simplicity, CIDDS excludes the PoW, we might need to add a PoW mechanism in the future. IOTA implements this mechanism by using Minimum Weight Magnitude (MWM) [47]. This checks for the number of trailing zeros in transaction hash.

To implement a PoW, we need to perform a brute force operation on the transaction hash to find a nonce with the correct number of trailing zeros. The user can implement this by themselves by modifying the base of the simulator to include a hashing algorithm. But this would require some more development effort, but would follow the same steps listed in subsection 5.4.1.

5.4.3 Result

Based on the case studies mentioned above, we could conclude that CIDDS is extensible and individual users are free to modify the opensourced code as they wish to extend CIDDS.

5.5 Visuals

CIDDS runs on the browser and uses Django Templates to render the visuals for the user. All the images available in the evaluation (unless specified otherwise) are directly taken from CIDDS and the user running the simulation could use them for intuition.

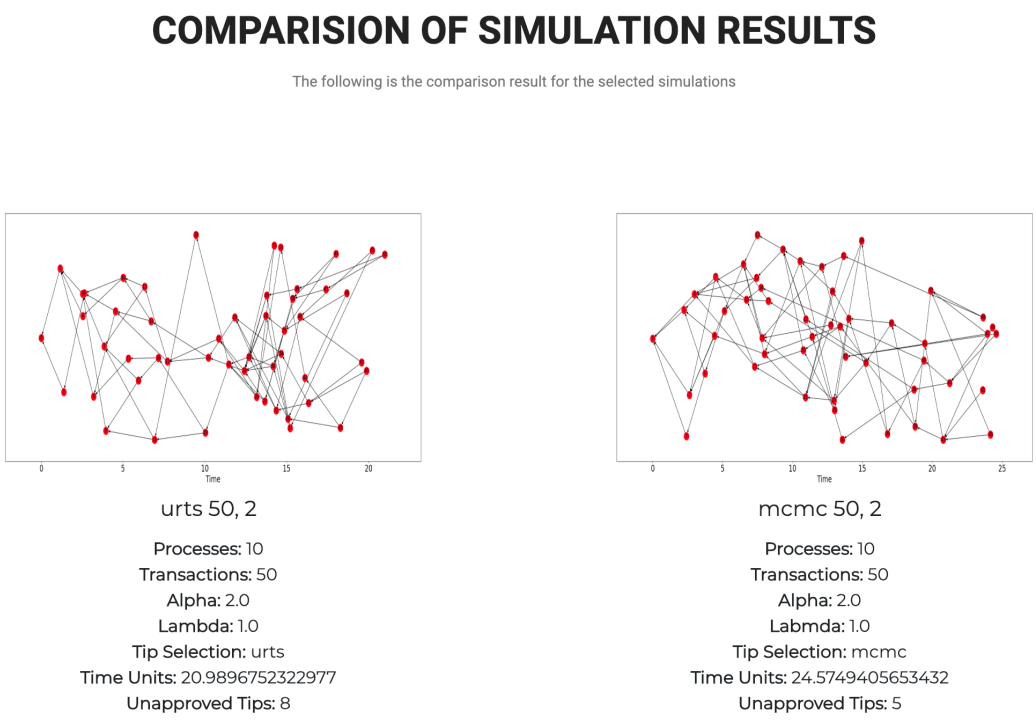


Figure 5.7: CIDDS - Comparison Visual

Also, CIDDS supports a comparison visual as shown in fig. 5.7, which makes it easier for the user to check simulations side by side fig. 5.7. One major limitation of CIDDS is the inability to accurately depict the visuals for simulations with more than 500 transaction. This would require a separate UI application to be built on top of CIDDS and this is discussed in section 6.3.

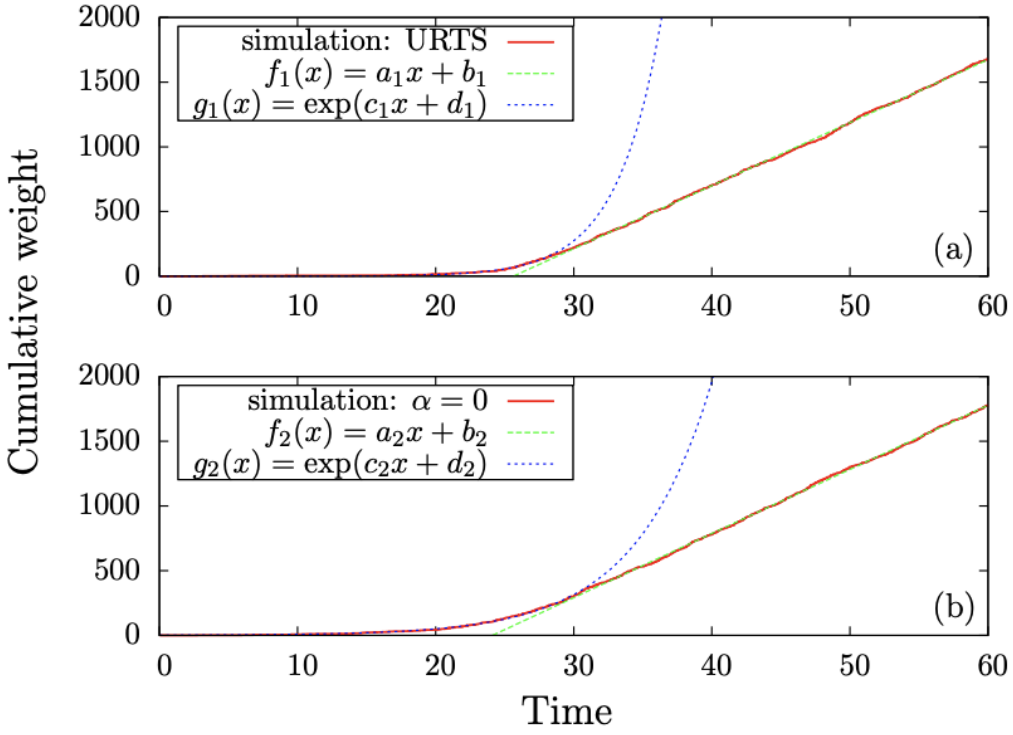


Figure 5.8: IOTA Simulation- Cumulative weights of the 200th transactions

Cumulative weights of the 200th transactions issued during the simulations of the Tangle for: (a) - tip selection URTS; (b) - tip selection MCMC with $\alpha = 0.001$. Flow rate of new transactions: $\lambda = 50$, each transaction approves exactly $k = 2$ other transactions.

5.6 Comparison with IOTA's Simulation results

In this section, we would try to replicate some of the experiments which IOTA published in [48]. Also, a note, the main focus of this thesis is not to attempt to replicate the results published by the IOTA, but to create a simulation framework, which could potentially used to replicate such results.

For replication of the result, a Python wrapper is written on top of CIDDS base to simulate the

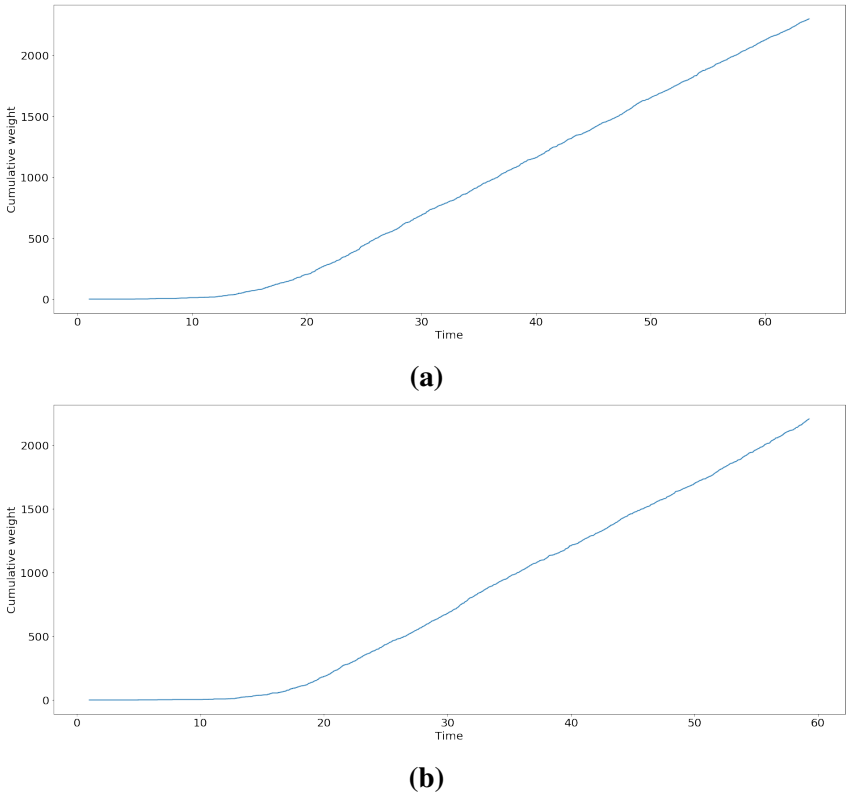


Figure 5.9: CIDDS Simulation - Cumulative weights of the 200th transactions

Cumulative weights of the 200th transactions issued during the simulations of the DAG for: (a) - tip selection URTS; (b) - tip selection MCMC with $\alpha = 0.001$. Flow rate of new transactions: $\lambda = 50$

conditions defined in the IOTA experiments [48].

We discuss a few aspects of [48] below:

5.6.1 Cumulative Weights on 200th Transactions

Comparison of cumulative weights of the 200th transaction is one of the first experiments in [48]. The results of this experiment is presented in fig. 5.8. The same experiment was run using CIDDS on Machine 1 and the result of these experiments are presented in fig. 5.9. The outcomes are pretty similar to the simulation from IOTA.

5.6.2 Number of Tips for Large Number of transactions

Another simulation from IOTO is a plot of the tip counts against the time with increase in number of transactions. This is displayed in fig. 5.10.

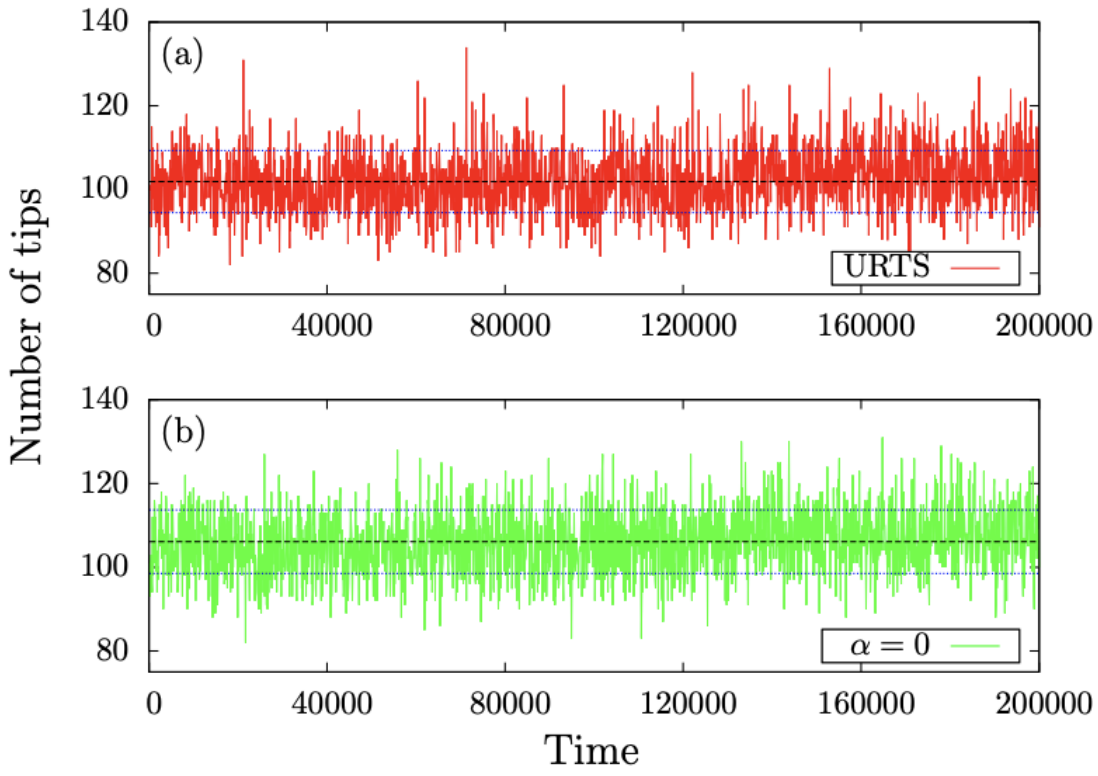


Figure 5.10: IOTA Simulation Tip Count

Values of L(t) - number of the tips (unconfirmed transactions) as a function of time for: (a) URTS; (b) tip selection MCMC $\alpha = 0$. Flow rate of new transactions: $\lambda = 50$; each transaction approves exactly $k = 2$ other transactions; simulation involves 10^7 transactions. L(t) varies greatly around average value (black long dashed curve) and most of the points are no more further from average than standard deviation (blue short dashed lines).

A similar simulation can be run by CIDDS. A simulation in Machine 2 with the parameters: $\lambda = 50$, URTS Tip Selection and number of transactions $n = 10^5$ is run and the results are presented in fig. 5.11. We do not perform a comprehensive recreation of the IOTA's simulations as the IOTA simulations as a part of this thesis. These are some of the sample demonstrations that CIDDS could be used to simulate large scale simulations and can be intuitive.

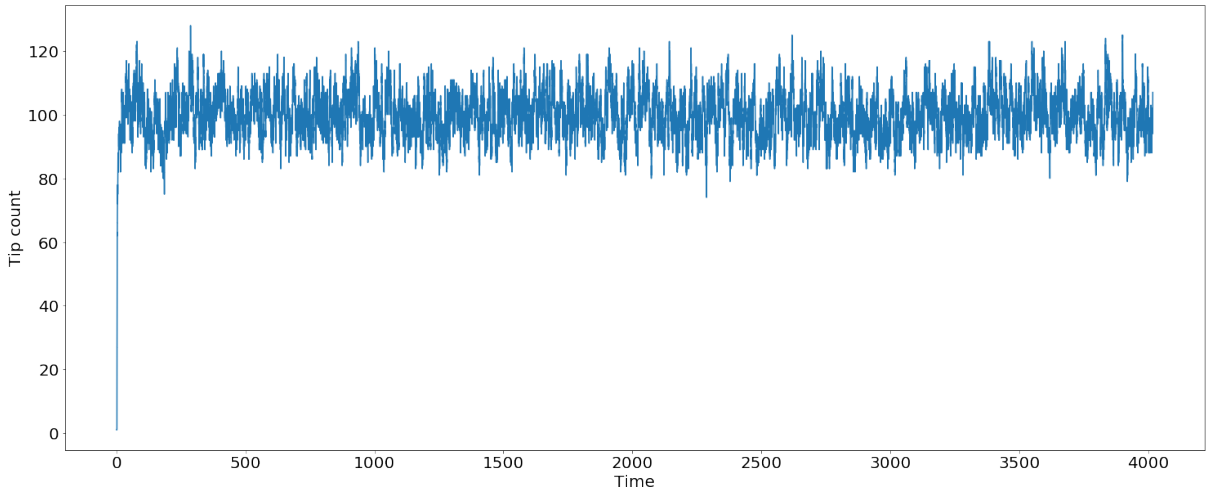


Figure 5.11: CIDDS Simulation - URTS Tip Count

5.7 Some interesting observations

5.7.1 Effect of transaction rates on the DAG

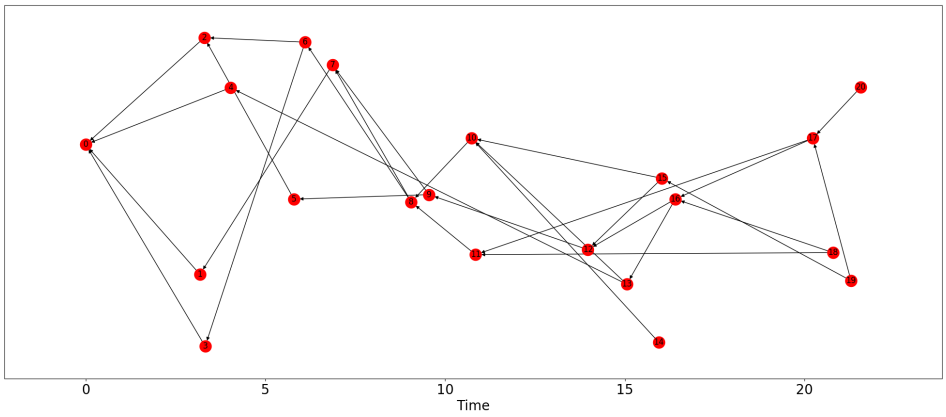


Figure 5.12: CIDDS Transaction Rate - Machine 2 - Simulation 1

One of the most apparent aspects which we can notice from the simulator is the effect of transaction rate on the time units. The total time units are inversely proportional to the transaction rates, i.e higher the transaction rate, lower the time units as discussed in the IOTA blog [49]. There are two simulations run on machine 2 with the following parameters:

- Simulation 1: n=20, α = 1, algorith=mcmc λ = 1

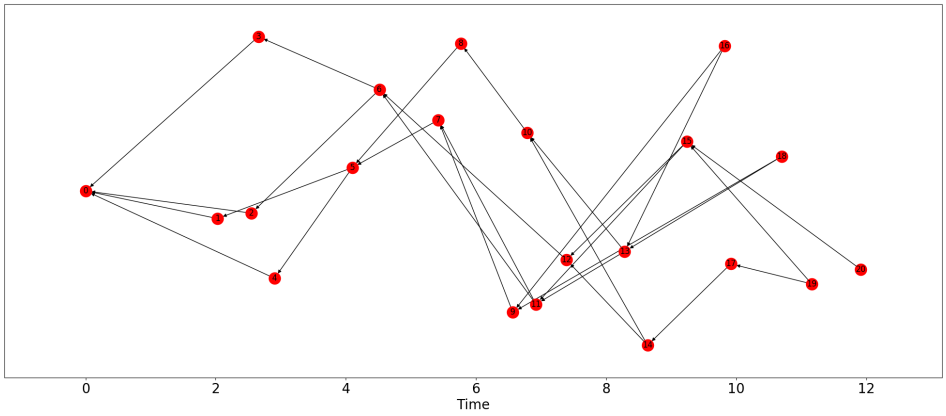


Figure 5.13: CIDDS Transaction Rate - Machine 2 - Simulation 2

- Simulation 2: n=20, $\alpha = 1$, algorith=mcmc $\lambda = 2$

From the figures, fig. 5.12 and fig. 5.13 it can be noticed that doubling the transaction rate while keeping all other parameters constant halves the time units taken.

5.7.2 Unapproved Tips in large simulations

Simulation	Transactions	Tip Selection	Time	Unapproved Tips
1	100000	urts	200141.99	9
2	200000	urts	390141.32	4
3	100000	mcmc	99897.82	194
4	200000	mcmc	174662.34	49

Table 5.2: CIDDS Evaluation - Unapproved Tips in Large simulations

One of the other interesting thing we noticed while simulating large simulations is that the number of unapproved tips remain very low for the number of transactions. For example consider the simulations with very large transactions done with parameters as described in table table 5.2. It can be noted that with increase in number of transactions, the unapproved tips do tend to decrease.

Many such interesting observations can be made by running more simulations with controlled parameters in CIDDS.

Chapter 6

Summary

For this thesis, we have designed and implemented a configurable and interactive simulator to simulate the DAG based distributed ledgers in large scale. CIDDS focuses on scale and performs fairly well with respect to speed even when executed with large scale simulations. Since there currently exist no public simulator at the time of writing this thesis to simulate the DAG based DLTs in large scale, CIDDS could be used as a valuable tool for researchers working with DAG based DLTs to study the network properties. We were also able to achieve parallel processing of the nodes to simulate how real world IOT devices might work.

In addition, we have developed a web application wrapper for the simulator to visualize and compare the DAGs generated and obtain a brief overview of the simulation results. Also, CIDDS can be extended further to integrate specific use cases as it is open sourced and free to use for anyone planning to research on DAG based DLTs. The design decisions followed an iterative approach of prototyping and validating the results in multiple cycles.

6.1 Status

We were able to achieve the goal of creating a simulator for generating large scale DAG based distributed ledgers with the user being able to study the properties of DAG closely by modifying individual parameters. CIDDS also stands as a strong base upon which extended applications could be created as described in section 5.4.

This thesis work was demoed at the Middleware Conference 2018 [50].

6.2 Conclusions

This thesis was both an interesting and challenging to work upon. With the gaining prominence of Distributed Ledger Technologies and their potential usages in multiple fields, it was very interesting to read about the technologies around this topic and implement this simulator. One of the important challenges faced is the lack of resources to base this thesis upon. Unlike a standard scientific work, where a scholar reads reputed published papers and gains a good idea of the field before working on the project, I had to resort to highly unconventional methods like getting in touch with the IOTA developers directly via Discord and other possible channels. Also since there are no other public simulation frameworks available, realistic simulation frameworks like CIDDS would greatly benefit the scientific community in building better DAG based DLTs.

6.3 Future Work

In this section, we discuss the potential enhancements on CIDDS.

6.3.1 Enhancement in Core Algorithms

Though CIDDS implements parallel processing, achieving true parallelism with respect to utilizing all the cores of the machine upon which it runs would be vital in enhancing the speed of execution exponentially. An ideal approach would be creating multiple threads while utilizing all the cores of the underlying machine to create a very efficient method to generate the DAG. This requires more investigation and would be very effective in making CIDDS considerably faster.

6.3.2 Enhancement of the parameters

CIDDS could be potentially enhanced to include all the parameters which IOTA provides, as explained in the case studies around extensibility of CIDDS.

6.3.3 Enhancement of the User Interface

One of the major limitations CIDDS faces is its inability to visually present the DAG to the user when the size is larger than 500 nodes. Though the user can still obtain the text representation

of the DAG in python pickle format [51]. This text format can be visually represented using modern UI technologies like D3.js [52] adding interactivity for the user.

Visual representation would help in better deductions of simulation results and enhanced interactivity would definitely help in noticing some very interesting properties of the DAG.

6.3.4 Enhancement of Comparisons

Currently CIDDS is limited to providing comparison of only 2 selected simulation results. This behaviour could be further investigated and we could come up with some interesting ways of presenting results of multiple simulations to the user for comparison.

6.3.5 Inclusion of other DAG based Distributed Ledgers

Right now, CIDDS focuses only on implementation of the DAG, which is named as Tangle, as proposed by IOTA. This can be further enhanced to include other interesting DAG based simulation frameworks like Hedera Hashgraph [10].

6.3.6 Attack Models on DAG based Distributed Ledgers

Another interesting enhancement to CIDDS would be to generate attacker models upon DAGs and analyze attack patterns. Some of the very interesting attacks which can be performed in a DAG based ledger and which might potentially be simulated by enhanced CIDDS are:

- **Double Spending**: A Parasite Chain attack can be modelled to perform double spending on a DAG where the attacker creates a transaction in main DAG, but could potentially create a conflicting transaction in a sub-chain, also dubbed as a parasite chain[53].
- **34% Attack**: The tangle whitepaper[1] states that if a single entity could control 34% of the total network, they can theoretically issue a malicious transaction. This is an interesting behaviour to study with a simulator.
- **Sybil attacks**: Flooding of the network with forged identities and performing Sybil attacks on the DAG network needs further investigation.

There are also many other possible attack scenarios which can be modelled and tried upon CIDDS by extending it.

6.3.7 Running more comprehensive Simulations

More simulations with controlled parameters can be run using CIDDS to closely observe the behaviour of DAGs. It can be a very valuable application of CIDDS. There are only a few sample simulations run and presented in chapter 5.

6.3.8 Tests

CIDDS does implement a few unit tests to validate the functionality (in tests.py). But further extensive regression tests are required.

These are some of the enhancements we propose for the future.

Appendices

Appendix A

Appendix

A.1 Source Code

Available at https://github.com/i13-msrg/cidds

A.2 Code Snippets

A.2.1 Django Settings Module

```
"""
Django settings for CIDDS project.

Generated by 'django-admin startproject' using Django 2.1.2.

For more information on this file, see
https://docs.djangoproject.com/en/2.1/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/2.1/ref/settings/
"""

import os

# Build paths inside the project like this: os.path.join(BASE_DIR, ...)
BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
```

```
# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/2.1/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = '***HIDDEN****'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = False

ALLOWED_HOSTS = ['0.0.0.0', '131.159.52.52', '.blockchainsimulator.org']

# ALLOWED_HOSTS = ['131.159.52.52','.blockchainsimulator.org' ]


# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'simulator.apps.SimulatorConfig',
    'django_tables2',
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'CIDDS.urls'

TEMPLATES = [
    {
```

```
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates')]
        ,
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'CIDDS.wsgi.application'


# Database
# https://docs.djangoproject.com/en/2.1/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}


# Password validation
# https://docs.djangoproject.com/en/2.1/ref/settings/#auth-password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'django.contrib.auth.
        password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
```

```
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]


# Internationalization
# https://docs.djangoproject.com/en/2.1/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_L10N = True

USE_TZ = True


# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/2.1/howto/static-files/

STATIC_URL = '/static/'

STATICFILES_DIRS = [
    os.path.join(BASE_DIR, "static/"),
]

STATIC_ROOT = '/static/'

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': "cidds",
        'USER': "postgres",
        'PASSWORD': ***HIDDEN***,
        'PORT': '5432',
        'HOST': 'localhost',
    }
}

DJANGO_TABLES2_TEMPLATE = 'django_tables2/bootstrap-responsive.html'
```

```
MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, "media/")

LOGIN_REDIRECT_URL = '/'
LOGOUT_REDIRECT_URL = '/'
```

Listing A.1: Django Settings

A.2.2 Source code for persistence model

```
from django.contrib.auth.models import User
from django.db import models
from django.db.models import Model
from django.utils import timezone
from safedelete.models import SafeDeleteModel
import uuid
import os

def get_image_path(instance, filename):
    return os.path.join('simulation_results', str(instance.id), filename)

class SimulationResults(Model):

    # User who initiated the error
    user = models.ForeignKey(
        User,
        null=True,
        editable=False,
        on_delete=models.PROTECT,
        verbose_name="Created by user",
        help_text="User who created the simulation"
    )
    # Number of processes
    num_process = models.IntegerField(blank=False, null=False, default= 1)

    # Number of transactions
    transactions = models.IntegerField(blank=False, null=False, default = 10)


    alpha = models.FloatField(blank=False, null=False, default=1)

# Degree of randomness
```

```
    randomness = models.FloatField(blank=False, null=False, default=1)

# Tip selection algorithm
    algorithm = models.CharField(blank=True, max_length=10,
                            help_text="A short description of the simulation")

    # reference text for the simulation
    reference = models.TextField(blank=True,
                              help_text= "A short description of the simulation" )

    # Picked version of the entire tangle
    dag = models.TextField(blank=True,
                        help_text= "The tangle result from the simulation")

    # Current status of the simulation - Can be running or done
    status = models.CharField(blank=True, max_length=20,
                            help_text="Status of the simulation")


    # resultant plotted image
    image = models.ImageField(upload_to='result_images', blank=True, null=True)

    # Created time
    created = models.DateTimeField(editable=False, default=timezone.now)

    # modified time
    modified = models.DateTimeField(null=True)

    # Time units
    time_units = models.FloatField(blank=True, null=True, default=0)

    # unapproved tips
    unapproved_tips = models.IntegerField(blank=True, null=True, default = 0)


    def save(self, *args, **kwargs):
        ''' On save, update timestamps '''
        if not self.id:
            self.created = timezone.now()
        self.modified = timezone.now()
        return super(SimulationResults, self).save(*args, **kwargs)
```

Listing A.2: Persistence Model of the Simulator

A.2.3 Source code for simulator urls.py

```
from django.urls import path
from django.views.generic import TemplateView


from simulator.views import StartSim, SimulationHistory, Comparison, Details

urlpatterns = [
    path('initialize/', StartSim.as_view(), name='initialize'),
    path('history/', SimulationHistory.as_view(), name='history'),
    path('start/', TemplateView.as_view(template_name='start_form.html'),
        name='startsim'),

    path('compare/', Comparison.as_view(), name='compare'),
    path('<int:sim_id>/', Details.as_view(), name='detail'),

]
```

Listing A.3: URL Routing in CIDDS

A.2.4 Source code for request handling - views.py

```
import io
from pprint import pprint

from django.contrib import messages
from django.core.files.base import ContentFile
from django.http import JsonResponse, HttpResponse
from django.shortcuts import render, redirect, render_to_response
from django.views import View
import json
from django.core.files.images import ImageFile

from django_tables2 import RequestConfig
from matplotlib.backends.backend_agg import FigureCanvasAgg

from base import Orchestrator
from django.http import HttpResponse

from simulator.models import SimulationResults
from simulator.tables import SimulationResultsTable
```

```
def handler404(request, template_name="error.html"):
    '''
    Handler for Page not found
    :param request: http request
    :param template_name: error template
    :return: response
    '''
    response = render_to_response("error.html")
    response.status_code = 404
    return response

def handler500(request, template_name="error.html"):
    '''
    Handler for internal server error
       :param request: http request
       :param template_name: error template
       :return: response
       '''

    response = render_to_response("error.html")
    response.status_code = 500
    return response


class StartSim(View):

    def get(self, request):
        if request.user.is_anonymous:
            user_notif = 'Please login. If you dont have ' \
                         'a login yet, please request access!'
            data = {
                'message': user_notif

            }
            return render(request, "default.html", data)

        nodes = request.GET.get('nodes')
        # processes = request.GET.get('processes')
        alpha = request.GET.get('alpha')
        randomness = request.GET.get('randomness')
```

```
        plot = Orchestrator.start_helper()
        buf = io.BytesIO()
        # plot.show()
        plot.savefig(buf, format="png")
        plot.show()
        response = HttpResponse(buf.getvalue(),content_type="image/png")
        # create your image as usual, e.g. pylab.plot(...)
        return response

    def post(self, request):

        if request.user.is_anonymous:
            user_notif = 'Please login. If you dont have ' \
                         'a login yet, please request access!'
            data = {
                'message': user_notif

            }
            return render(request, "default.html", data)

        data = request.POST
        nodes = int(data.get("transactions"))
        processes = int(data.get("processes"))
        alpha = float(data.get("alpha"))
        randomness = float(data.get("randomness"))
        algorithm = data.get("algorithm")
        reference = data.get("reference")
        pprint(data)
        sim = SimulationResults(user=request.user,
                                num_process=processes,
                                alpha=alpha,
                                randomness=randomness,
                                reference=reference,
                                algorithm=algorithm,
                            transactions=nodes
                                )
        sim.status = "Running"
        sim.save()

        id = sim.id

        t = Orchestrator.start_helper(sim)
        figure = io.BytesIO()
```

```
        # plot.show()
        plot = t.plot()
        plot.savefig(figure, format="png")
        plot.show()


        sim = SimulationResults.objects.get(id=id)
        sim.status = "Done"

        resultImage = ImageFile(figure)
        sim.image.save(str(id)+'.png',resultImage)
        sim.tangle = t
        sim.reference = reference
        sim.unapproved_tips = len(t.tips())
        sim.time_units = t.time
        sim.save()

        table_results = SimulationResultsTable(
            SimulationResults.objects.all(),order_by="-created")
        RequestConfig(request).configure(table_results)
        pprint("*****************************************************************" )

        pprint(table_results )
        data = {
            'Title': 'Simulation History',
            'table': table_results,
            'messages': messages
        }
        return render(request, "simulation_results.html", data)

        # response = HttpResponse(figure.getvalue(), content_type="image/png")

        # return response


class SimulationHistory(View):

    def get(self, request):
        if request.user.is_anonymous:
            user_notif = 'Please login. ' \
                         'If you dont have a login yet, please request access!'
            data = {
                'message': user_notif
```

```
            }
            return render(request, "default.html", data)

        pprint("*****************************************************************" )
        pprint("simulation history")

        table_results = SimulationResultsTable(
            SimulationResults.objects.all(),order_by="-created")
        RequestConfig(request).configure(table_results)
        pprint(table_results)


        data = {
            'Title': 'Simulation History',
            'table': table_results,
            'messages': messages

        }

        pprint(data)

        return render(request, "simulation_results.html", data)



class Comparison(View):

    def post(self, request):
        data = request.POST
        pks = data.getlist("sim_selection")
        selected_objects = SimulationResults.objects.filter(pk__in=pks)

        data = {
            'first_sim': selected_objects[0],
            'second_sim': selected_objects[1]

        }

        return render(request, "compare.html", data)

class Details(View):
```

```
    def get(self, request, sim_id):
        # id = int(request.GET.get('sim_id'))
        selected_object = SimulationResults.objects.get(id=sim_id)

        data = {
            'sim': selected_object,

        }

        return render(request, "details.html", data)
```

Listing A.4: Handling Routing in CIDDS - views.py

```
import django_tables2 as tables
from django.utils.html import format_html
from django_tables2 import A

from simulator.models import SimulationResults


class CheckBoxColumnWithName(tables.CheckBoxColumn):
    @property
    def header(self):
        return self.verbose_name

class InteractiveLink(tables.Column):
    url = '<a href="http://localhost:8080/?id={id}">Interactive</a>'

    def render(self, record):
        return format_html(self.url, id=record.simulator.id,
                           title=record.title)

class SimulationResultsTable(tables.Table):
    sim_selection = CheckBoxColumnWithName(
        verbose_name="Slection", accessor="pk",
        attrs={"th__input":

        {
          "onclick": "toggle(this)"}},
          orderable=False)
    interactive = InteractiveLink()

    details = tables.LinkColumn(
```

```
        'detail', text='Details', args=[A('pk')], empty_values=())
    class Meta:
        model = SimulationResults
        fields = ('sim_selection','id','details',
                'reference' , 'algorithm', 'status',
                'image', 'interactive', 'created', 'modified')
```

Listing A.5: Handling Routing in CIDDS - views.py

A.3 CIDDS Evaluation - Speed

Transactions(n)	Time(s)
10	1.2
20	2.5
50	4.5
100	34
200	64
500	129
1000	304
5000	502
10000	1032
50000	4427
100000	7311
150000	10890
200000	15589
250000	19304
300000	22898

Table A.1: Transactions vs Time on Machine 1

Transactions(n)	Time(s)
10	0.03
20	0.04
50	4.5
100	8
200	11
500	43
1000	136
5000	231
10000	832
50000	1345
100000	3144
150000	7635
200000	9434
250000	10034
300000	14245

Table A.2: Transactions vs Time on Machine 2

Bibliography

[1] S. Popov. (2016, April) The tangle. [Online]. Available: http://www.tangleblog.com/wp-content/uploads/2016/11/IOTA_Whitepaper.pdf

[2] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," http://bitcoin.org/bitcoin.pdf."

[3] (2018, December) Etherium foundation. [Online]. Available: https://www.ethereum.org/

[4] (2018, December) Bitcoin energy consumption. [Online]. Available: https://digiconomist.net/bitcoin-energy-consumption

[5] R. M. R. Yasaweerasinghelage, M. Staples, and I. Weber, "Predicting latency of blockchain-based systems using architectural modelling and simulation," 04 2017.

[6] (2018, December) Blockchain explorer. [Online]. Available: https://blockexplorer.com/

[7] L. Stoykov, K. Zhang, and H.-a. Jacobsen, "Vibes: fast blockchain simulations for large-scale peer-to-peer networks: demo," 12 2017, pp. 19–20.

[8] (2018, December) Cryptocurrency market capital. [Online]. Available: https://coinmarketcap.com/

[9] (2017, November) The first glance at the simulation of the tangle: discrete model. [Online]. Available: https://assets.ctfassets.net/r1dr6vzfxhev/2ZO5XxwehymSMsgusUE6YG/f15f4571500a64b7741963df5312c7e7/The_First_Glance_of_the_Simulation_Tangle_-_Discrete_Model_v0.1.pdf

[10] (2018, December) Hedera hashgraph. [Online]. Available: https://www.hedera.com/

[11] *Django documentation*, https://docs.djangoproject.com/en/1.10/, last visited: 12/09/2018. [Online]. Available: https://docs.djangoproject.com/en/1.10/

[12] K. S. L. L. Deshpande, Advait and S. Gunashekar, "Understanding the landscape of distributed ledger technologies/blockchain: Challenges, opportunities, and the prospects for standards," *BSI*, 2017.

[13] (2018, December) Cryptokitties. [Online]. Available: https://www.cryptokitties.co/

[14] (2018, December) Tradix. [Online]. Available: https://tradeix.com/distributed-ledger-technology/

[15] e. a. Crosby, Nachiappan, "Sutardja center for entrepreneurship and technology technical report," *Sutardja Center for Entrepreneurship and Technology Technical Report*, 2015.

[16] (2018, December) Blockchain wikipedia entry. [Online]. Available: https://en.wikipedia.org/wiki/Blockchain

[17] R. Schollmeier, "A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications," in *Proc. of the First International Conference on Peer-to-Peer Computing*, 2011, p. 101 – 102.

[18] D. H. C. I. W. L. J. S. S. S. H. S. D. T. D. Demers, Alan; Greene, "Epidemic algorithms for replicated database maintenance. proceedings of the sixth annual acm symposium on principles of distributed computing," 1 1987, pp. 1–12.

[19] (2018, December) Lisk academy - consensus protocols. [Online]. Available: https://lisk.io/academy/blockchain-basics/how-does-blockchain-work/consensus-protocols

[20] H.-a. Jacobsen, M. Sadoghi, M. Hossein Tabatabaei, R. Vitenberg, and K. Zhang, "Blockchain landscape and ai renaissance: The bright path forward," 12 2018, pp. 1–1.

[21] (2018, December) Bitcoin wiki. [Online]. Available: https://en.bitcoin.it/wiki/Transaction

[22] (2018, December) Iota market capitalization. [Online]. Available: https://coinmarketcap.com/currencies/iota/

[23] B. Breier, "Technical analysis of the tangle in the iota-environment - bachelor thesis in informatics, technical university of munich," 2017. [Online]. Available: https://wwwmatthes.in.tum.de/pages/j4kcsh9lbjby/Bachelor-s-Thesis-of-Bennet-Breier

[24] (2018, December) Iota - anatomy of a transaction. [Online]. Available: https://iota.readme.io/docs/the-anatomy-of-a-transaction

[25] (2018, December) Emulation. [Online]. Available: https://whatis.techtarget.com/definition/emulation

[26] J. B. J. C. B. N. D. Nicol, *Discrete-Event System Simulation*. Prentice Hall.

[27] A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, and S. Capkun, "On the security and performance of proof of work blockchains," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*,

ser. CCS '16. New York, NY, USA: ACM, 2016, pp. 3–16. [Online]. Available: http://doi.acm.org/10.1145/2976749.2978341

[28] (2018, December) Ns3. [Online]. Available: https://www.nsnam.org/

[29] (2018, December) Akks. [Online]. Available: https://akka.io/

[30] A. Gal. (2018, August) Tangle simulation. [Online]. Available: https://simulation1.tangle.works/

[31] M.-N. Nguyen. (2018, June) Tangle simulation. [Online]. Available: https://github.com/minh-nghia/TangleSimulator

[32] (2018, December) Jupyter notebook. [Online]. Available: https://jupyter.org/

[33] (2018, December) Iota discord channel. [Online]. Available: https://discordapp.com/invite/fNGZXvh

[34] (2018, December) Alon gal, iota foundation. [Online]. Available: https://blog.iota.org/@alon.gal

[35] S. N. C. D. S. W. S. K. J. Mecke, *Stochastic Geometry and Its Applications*. John Wiley and Sons.

[36] (2018, December) Poisson point process. [Online]. Available: https://en.wikipedia.org/wiki/Poisson_point_process

[37] (2018, December) ngnix. [Online]. Available: https://www.nginx.com/

[38] (2018, December) Postgressql. [Online]. Available: https://www.postgresql.org/

[39] C. Larman and V. R. Basili, "Iterative and incremental development: A brief history," *Computer*, vol. 36, pp. 47–56, 06 2003. [Online]. Available: doi.ieeecomputersociety.org/10.1109/MC.2003.1204375

[40] (2018, December) Networkx. [Online]. Available: https://networkx.github.io/

[41] (2018, December) Python pathos. [Online]. Available: https://pypi.org/project/pathos/

[42] (2018, December) Django sites. [Online]. Available: https://djangostars.com/blog/10-popular-sites-made-on-django/

[43] (2018, December) Django advantages. [Online]. Available: https://www.djangoproject.com/start/overview/

[44] (2018, December) Standard django project. [Online]. Available: https://docs.djangoproject.com/en/2.1/intro/tutorial01/

[45] (2018, December) Django templates. [Online]. Available: https://docs.djangoproject.com/en/2.1/topics/templates/

[46] (2018, December) Plot.ly online plotter. [Online]. Available: https://plot.ly/create/

[47] (2018, December) Iota proof of work. [Online]. Available: https://docs.iota.org/introduction/tangle/proof-of-work

[48] (2018, May) Extracting tangle properties in continuous time via large-scale simulations. [Online]. Available: https://assets.ctfassets.net/r1dr6vzfxhev/4T4IAlxk9ym0eWco0UoQIQ/90094e746745b89253eb3636b4ad1597/Extracting_Tangle_Properties_in_Continuous_Time_via_Large_Scale_Simulations_V2.pdf

[49] A. Gal. (2018, December) Iota blog - tangle an illustrated introduction. [Online]. Available: https://blog.iota.org/the-tangle-an-illustrated-introduction-c0a86f994445

[50] M. R. A. Lathif, P. Nasirifard, and H.-A. Jacobsen, "Cidds: A configurable and distributed dag-based distributed ledger simulation framework," in *Proceedings of the 19th International Middleware Conference (Posters)*. ACM, 2018, pp. 7–8.

[51] (2018, December) Python pickle serialization. [Online]. Available: https://docs.python.org/3/library/pickle.html

[52] (2018, December) D3.js. [Online]. Available: https://d3js.org/

[53] (2018, December) Parasite chain attack on iota. [Online]. Available: https://blog.iota.org/attack-analysis-the-simple-parasite-chain-42a34bfeaf23