The Tangle

## ABSTRACT

As effective as blockchain may be its constraints such as scalability and miners are what lead to the innovation of IOTA, a unique new platform for distributed ledgers that implement the architecture known as Tangle. With the world flourishing with efficient IOT devices that generate prodigious amounts of data, the concern for security, storage and privacy is more relevant than ever. With IOTA’s Tangle, which at its core uses a directed acyclic graph for storage and transaction purposes removes the use of miners that demands a significant aggregate of processing power and makes the platform lightweight in nature. In this project, we study and implement IOTA’s data structure, the Tangle, areas where it has been used and most significantly the edge it has over blockchain.

## INTRODUCTION

Unlike the other cryptocurrencies available, The IOTA is unique in the sense that it isn’t built upon the traditional blockchain structure. The project developers have built this cryptocurrency on a data structure they’ve implemented from scratch, known as “The Tangle”. The aim was to create a distributed ledger which is cost effective and readily scalable with the onset of Internet of Things.

The key principle behind The Tangle is that of a directed acyclic graph. [1] The network contains several independent nodes without forming loops. Like the concept of “Proof of Work” in the traditional blockchain, The Tangle has its own efficient mechanism wherein every incoming transaction must validate two previous transactions with a small amount of work. The upside of this method is that it doesn’t include any extra fees to get the work done.

The Blockchain, as impressive as it is, has its own disadvantages. Some of which include scalability, capacity/storage, increasing bandwidth to keep transactions updated and high transaction fees. The Tangle, however, is essentially a big web which makes it easier to overcome the problems faced by the Blockchain. Another key advantage possessed by the Tangle is the higher throughput due to parallelized validation.

The tangle has many use cases at present, some of which include microtransactions, free transactions, bank transfer, machine to machine payments and Internet of Things.

## LITERATURE REVIEW

The Tangle, in simple terms is the data structure used in IOTA. It’s a form of a directed acyclic graph which is used to hold the transactions. Each node/vertex in the graph represents a transaction.

At the heart of the tangle, is the ‘genesis’, which is the initial node in the network and is ultimately verified directly/indirectly by all the subsequent nodes in the network. The genesis transaction then sends tokens to several founder addresses to essentially create the basis of the network. Whenever a new incoming transaction then joins the tangle, it must choose 2 existing transactions to approve and this was new edges are added to the graph. [2]

“For a node to issue a valid transaction, the node must solve a cryptographic puzzle like those in the Bitcoin blockchain. This is achieved by finding a nonce such that the hash of that nonce concatenated with some data from the approved transaction has a form. In the case of the Bitcoin protocol, the hash must have at least a predefined number of leading zeros.” [3]

If a node finds out a certain transaction is conflicting in the tangle’s history, then that particular transaction will not be approved. More the number of approvals for a transaction, higher is its confidence with the system.

Transactions which haven’t been approved yet are called tips, the key to tangle’s efficiency and effectiveness comes from the manner with which the tips are selected. Transactions which approve older transactions and aren’t up to date with the current transactions are called ‘lazy tips. [4] In order to avoid this manipulation of the network through these lazy tips, the concept of weights was introduced. The idea behind this is that the amount of work/approvals done by a transaction will result in a higher value of weight. The transaction with a higher value of weight is in way more important to the network and has higher probability to be selected for approval than a corresponding transaction with lower weight.

Once the concept of weights has been established, this is further used to calculate the cumulative weights of each node. The cumulative weight is defined as the summation of weight of a node with the weights of all other nodes that have approved this node directly or indirectly.

The cumulative weights form the basis for consensus and confirmation of transactions. The more the cumulative weight of a transaction, the higher is the confidence with which it is accepted by the system. [5]

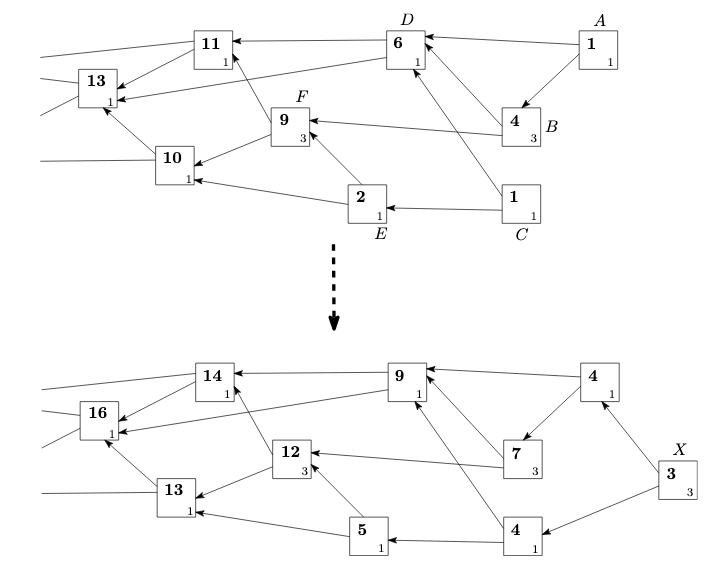


Figure 1: DAG with weight assignments before and after a newly issued transaction X. The boxes represent transactions, the small number in the corner of each box denotes own weight, and the bold number denotes the cumulative weight.

As we can see in Figure 1, Transaction F is being approved by A, B, C, E directly/indirectly. Therefore, it has a cumulative weight of 9, which is equal to the sum of its weight and the weights of the other four transactions (3+1+1+3+1)

In order to proceed to the to the approval algorithms, two important terminologies related to the transactions have been defined, namely, height and depth. The height is defined as “the longest path of a node to the genesis” and the depth is defined as “the length of the longest reverse-oriented path to some tip.” This leads us to the next feature of a transaction, known as the score. The score of a transaction is defined as the summation of its own weight with the weights of all the transactions it has approved.

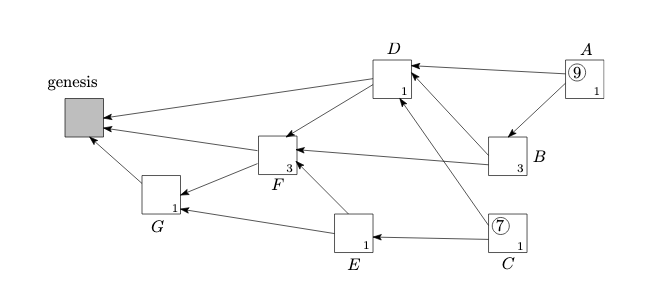


Figure 2DAG with own weights assigned to each site, and scores calculated for sites A and C.

As observed in Figure 2, Transaction a directly/ indirectly approves nodes B, D, F, G. Therefore, it has a score of 9 (1 + 1 + 3 + 3 + 1). Similarly, Transaction C has a score of 7.

**Algorithm**

**First, we define an algorithm that guides the working of individual nodes.**

*//Generate an anonymous wallet by generating a pair of public and private keys with a 2048-bit moduli in hex representation*

private\_key, public\_key = wallet\_generator()

*//Nodes with intention to participate in the process or to issue a new transaction run tip selection algorithms to choose tips to be approved, this offers attaching sites to their new transaction*

tips = find\_tips()

*//these new tips should be valid themselves (i.e. the transaction needs have a valid signature of the sender)*

*In case conflicting transactions are found the tip, selection algorithm is run again to find new tips*

If verify\_signature(tips [0]) and verify\_signature(tips [1]) is true:

return tips

else:

tips = find\_tips ()

*//If the selected tips are valid and do not point to conflicting transactions, we need to perform proof of work by finding nonce such that when appended to the approved transactions the hash of this combination start with a certain number of zeros that depend on pre decided difficulty*

nonces = proof\_of\_work(tips)

*//The new transaction details are now generated and signed. The transaction unit contains various elements from the current and previous transaction*

new\_transaction = Transaction\_block {Sender Address, Recipient Address,

Amount, Signature,

Nonces, Previous Hashes,

Timestamp and Cumulative Weight}

*//The new block is now appended to the tangle graph. This step adds edge from new transaction to the approved transactions and trickles down an update of cumulative weights. In practice it is broadcasted to the neighbours to be appended to their copy of the tangle too*

Tangle.add(new\_transaction)

Tangle.update\_edges(new\_transaction, tips)

Tangle.update\_cumulative\_weights(new\_transaction, tips)

**Secondly, we define an algorithm that guides the working of individual modules used above.**

The nodes are free to choose any tip selection algorithm, but we will see how choosing an algorithm (Weighted Random Walk) is inherently incentivised

Tip Selection (Find Tips) – Provides 4 Tip selection algorithm

1. **Random Tip Selection**

Selects any tip randomly with equal probability

1. **Recently Added K tips**

Selects two tips from the newly generated K tips

K can be chosen by the node

1. **Unweighted Random Walk**

STEP 0: Perform STEPS 1 to 3 many times, ~100 times after which go to STEP 4

STEP 1: Start with genesis as current node

STEP 2: Move to a random node that approves current node

STEP 3: Check if this is a tip

if current node is a tip:

Increment count of this tip

else:

Go to STEP 2

STEP 4: return two tips with maximum counts

1. **Weighted Random Walk**

STEP 0: Perform STEPS 1 to 4 many times, ~100 times after which go to STEP 5

STEP 1: Start with genesis as current node

STEP 2: Define Probability of the nodes according to their current cumulative weight

STEP 3: Jump to a random node according to these probabilities

STEP 4: Check if this is a tip

if current node is a tip:

Increment count of this tip

else:

Go to STEP 2

STEP 5: return two tips with maximum counts

*Yes*

*No*

*Selecting Attachment Sites and Validating Transactions*

Wallet

Run Tip Selection Algorithm

Validate Selected Transactions

Validate Chain of Transactions Approved by chosen transaction

Approve Tips

Create Own Transaction Data

Sign Transaction

Perform Proof of Work

Append to Tangle

*Creating Own Transaction*

Figure 3 Flowchart of the process

**Code Snippets of Implementation**

1. **Wallet Generator**
2. def generate\_wallet():
3. random\_gen = Crypto.Random.new().read
4. private\_key = RSA.generate(2048, random\_gen)
5. public\_key = private\_key.publickey()
6. response = {
7. 'private\_key': binascii.hexlify(private\_key.export\_key(format('PEM'))).decode('ascii'),
8. 'public\_key': binascii.hexlify(public\_key.export\_key(format('PEM'))).decode('ascii')
9. }
10. WALLET\_LIST.append((public\_key, private\_key))
11. return response
12. **Signature Generator**
13. def sign(sender\_private\_key, transaction\_dict):
14. private\_key\_obj = RSA.importKey(binascii.unhexlify(sender\_private\_key))
15. signer\_obj = PKCS1\_v1\_5.new(private\_key\_obj)
16. hash\_obj = SHA256.new(str(transaction\_dict).encode('utf-8'))
17. return binascii.hexlify(signer\_obj.sign(hash\_obj)).decode('ascii')
18. **Signature Verifier**
19. def verify\_signature(sender\_public\_key, signature, transaction):
20. public\_key = RSA.importKey(binascii.unhexlify(sender\_public\_key))
21. verifier = PKCS1\_v1\_5.new(public\_key)
22. h = SHA256.new(str(transaction).encode('utf-8'))
23. try:
24. verifier.verify(h, binascii.unhexlify(signature))
25. return True
26. except ValueError:
27. return False

1. **Proof of Work**
2. def valid\_proof(previous\_hashes, transaction\_dict, nonce):
3. guess = (str(previous\_hashes)+ str(transaction\_dict) + str(nonce)).encode('utf-8')
4. h = hashlib.new('sha256')
5. h.update(guess)
6. guess\_hash = h.hexdigest()
7. return guess\_hash[:DIFFICULTY] == '0'\*DIFFICULTY

10. def proof\_of\_work(previous\_hashes, transaction\_dict):
11. nonce = 0
12. while not valid\_proof(previous\_hashes, transaction\_dict, nonce):
13. nonce = nonce + 1
15. return nonce
16. **Weighted Random Walk**
17. def random\_walk\_weighted(self, current\_node=GENESIS\_ID):
18. if len(self.reverse\_edges[current\_node]) == 0:
19. return current\_node
21. elif len(self.reverse\_edges[current\_node]) < 3:
22. option = np.random.choice(np.arange(0,2))
23. if option==0:
24. return current\_node
26. prob = []
27. for next\_node in self.reverse\_edges[current\_node]:
28. prob.append(self.transactions[next\_node].cumulative\_weight)
30. prob = prob/np.sum(prob)
32. choice = np.random.choice(np.arange(0,len(self.reverse\_edges[current\_node])), p=prob)
33. return self.random\_walk\_weighted(self.reverse\_edges[current\_node][choice])
34. **Unweighted Random Walk**
35. def random\_walk\_unweighted(self, current\_node=GENESIS\_ID):
36. if len(self.reverse\_edges[current\_node]) == 0:
37. return current\_node
39. elif len(self.reverse\_edges[current\_node]) < 3:
40. option = np.random.choice(np.arange(0,2))
41. if option==0:
42. return current\_node
44. choice = np.random.choice(np.arange(0,len(self.reverse\_edges[current\_node])))
45. return self.random\_walk\_weighted(self.reverse\_edges[current\_node][choice])
46. **Recently Added Tip Selection**
47. return list(random.sample(set(list(self.transactions.keys())[-5:]), 2))
48. **Random Tip Selection**
49. return list(random.sample(set(self.transactions.keys()), 2))

**Hardware and Software Used**

|  |  |
| --- | --- |
| **Experimental Environments** | |
| **Hardware** | Generic Laptop |
| **Software** | Visual Studio Code  Python Compiler |

**Results**

*In the following Directed Acyclic Graphs* [6]

*Green sites represent unapproved transactions or tips*

*Blue sites represent transactions approved by at least one site*

*The letter represents ID and the numbers represent cumulative weight*

A 100

*For e.g. This is a site approved by at least one other site with ID ‘A’ and weight 100*

**A picture containing skiing, small, hanging, table

Description automatically generated**

Figure 4 Tangle Snapshot on using Random Tip Selection Algorithm

1. Random tip selection algorithm provides promising results, however, as can be see in the snapshot of the tangle above nodes may not approve new transactions. Such nodes are called lazy nodes. Random selection does not penalise this behaviour rather it in a sense incentives it as it allows nodes to approve the same old transactions and thus avoid doing proof of work, which is essential for tangles stability.

**A close up of a necklace

Description automatically generated**

Figure 5 Tangle Snapshot on choosing recently added tips for approval, K = 2 (Chooses from top 2 transactions)

**A picture containing accessory, small, necklace, bear

Description automatically generated**

Figure 6 Tangle Snapshot on choosing recently added tips for approval, K = 5 (Chooses from top 5 transactions)

1. In order to encourage new transaction approvals, we do random selection from top few tips on the tangle. With smaller K we get a linear (block chain) like chain of transactions which will not work in high load and leave the tangle with split ends. We increase the number of K and get the desired behaviour from the tangle. We learn that choosing new transactions gives us desired tangle properties. Having said this, we need a mechanism to end up at tips.

**A picture containing accessory, small, necklace, hanging

Description automatically generated**

Figure 7 Unweighted Random Walk Tip Selection Algorithm

1. The Markov Chain Monte Carlo Algorithms provide a mechanism to reach to the tips in order to select them. We perform a random walk from genesis towards the tips. We select tips that we end up with the greatest number of times on doing the random walks. However, forcing nodes to follow an algorithm is against the principles of decentralisation and this in no way stops the other nodes to be lazy. As can be seen we end up with the same problem of lazy nodes here. We see and increased number of non-approved tips. Which even though valid are orphaned. The nodes by selecting previous transactions would rather increase their chances of approval since they will have equal probability of getting selected compared to the complete set of all nodes that approve the same transaction.

**A necklace and blue hanging from a wire

Description automatically generated**

Figure 8 Weighted Random Walk Tip Selection Algorithm

1. Making the random walks weighted during the random walks in MCMC algorithms provide us with a solution, given that most of the nodes in the network are honest (which is presumption that all crypto currencies make). The walker selects nodes based on their cumulative weights and performs random walks. Now that tips can only have cumulative weights equal to their own weights, there chances of getting approved depends majorly on the validity of the transactions that they have approved.

The weighted random walks solve another problem, double spending. An attempt to double spend will create a fork in the tangle. With time the probability that new transactions will favour one fork will increase (exponentially), ultimately abandoning the other transaction.

In bonus it provides us with a way to decide if a transaction has been accepted by the block chain by allowing us to calculate confidence with which the transaction has been accepted by the graph. This is calculated as percent of times the particular site was encountered during a random walk.

**Conclusion**

The weighted Random walk provides us with a very efficient way of tip selection. Not following the proposed MCMC algorithm for mini computational gains does no good to the nodes and such lazy behaviour is discouraged by the other honest nodes. The added benefits of random walk in providing automatic solution to double spends and by providing a way to measure confidence of transaction approval makes it ideal for the scenario that IOTA aims to deal with.

# References

|  |  |
| --- | --- |
| [1] | A. Gal, "The Tangle: an Illustrated Introduction," [Online]. Available: https://blog.iota.org/the-tangle-an-illustrated-introduction-4d5eae6fe8d4. |
| [2] | A. Gal, "Visualisation," [Online]. Available: https://simulation1.tangle.works/. |
| [3] | S. Popov, "The Tangel," [Online]. Available: https://assets.ctfassets.net/r1dr6vzfxhev/2t4uxvsIqk0EUau6g2sw0g/45eae33637ca92f85dd9f4a3a218e1ec/iota1\_4\_3.pdf). |
| [4] | D. Liew, "Tip Selection in Tangle," [Online]. Available: https://www.blockchainguide.biz/tips-selection-in-tangle/. |
| [5] | Noneymous, "IOTA Transactions, Confirmation and Consensus," [Online]. Available: https://github.com/noneymous/iota-consensus-presentation. |
| [6] | Networkx, "Basic Networks from Pandas DataFrame," [Online]. Available: https://python-graph-gallery.com/320-basic-network-from-pandas-data-frame/. |