IOTA

a cryptocurrency for the Internet-of-Things (IoT) industry.

Real-life Examples of IOTA use

The main focus of the IOTA developers is on the Internet of Things (IOT) and M2M transactions.

- A smart cable that connects to a smart socket and without any human interaction pays for electricity.

- Your device will get connected to the internet and it will pay for the data in real-time as you browse the internet

Problem with Bitcoin

1. **Heavy fee**: As the amount of payment reduces, the relative fee rate rises.
2. **Separation of roles:** Bitcoin requires two types of users: transaction verifier (miners) and transaction issuers (normal users). Lacking the homogeneity among roles, the chance of conflicts remains high, wasting everyone's resources to solve the conflicts.

Why IOTA

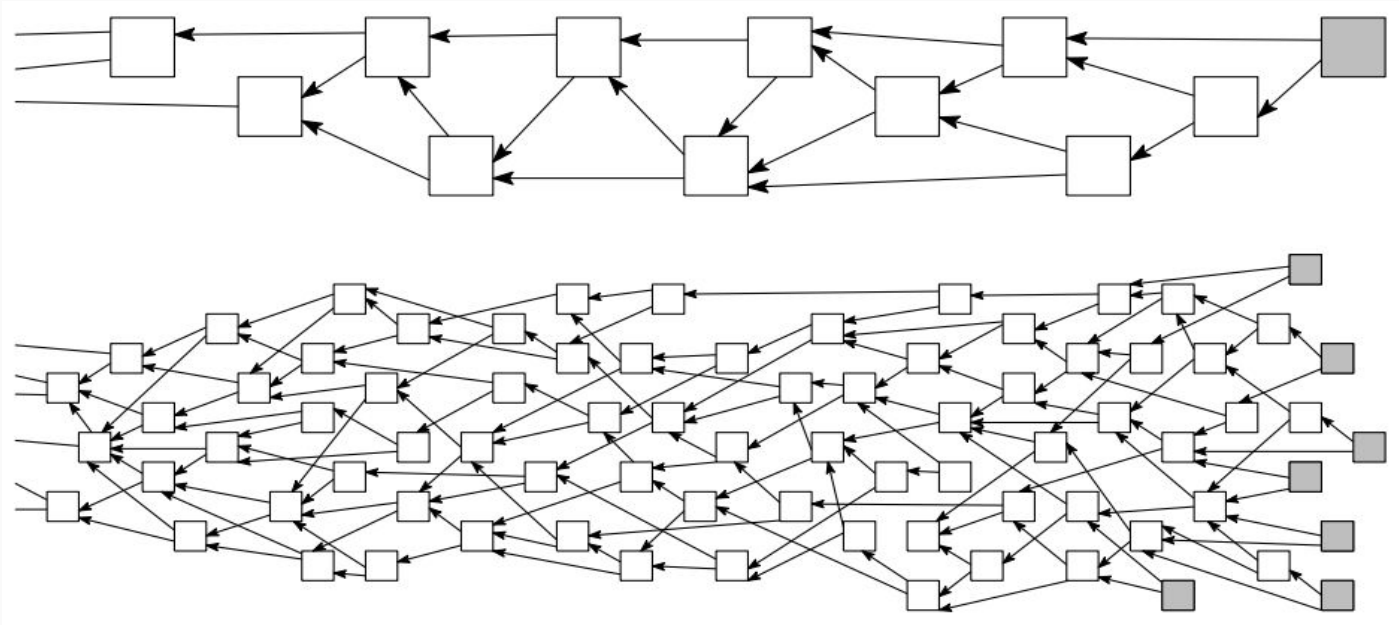
- Zero transaction costs

- Potential for indefinite scalability

Tangle

Tangle retain the blockchain features of the distributed ledger and secure transactions, but does not work with blocks

Uses Directed Acyclic Graph (DAG)

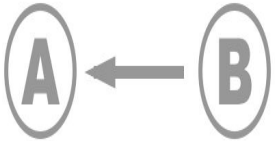


Tangle is made of sites and nodes:

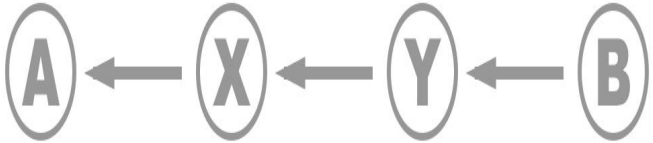
- **Sites:** Transactions represented on the graph.
- **Nodes:** Issuers of transactions.

Key Rule: *A newly issued transaction is obligated to approve* ***TWO*** *old transactions.*

NO MORE MINERs!



Direct Approval of Transaction



Indirect Approval of Transaction

Relationship between transactions

Who started up Tangle?

Genesis transaction is the first transaction in Tangle:

- It is directly/indirectly approved by all other transactions.
- It sends tokens from “an address containing all tokens” to other “founder addresses”
- To ensure equitable token distribution, a crowdsale was held from November to December 2015 during which 100% of the token supply was issued. Zero tokens were held for the developers/founders.
- The foundation currently holds just 5% of the total supply.

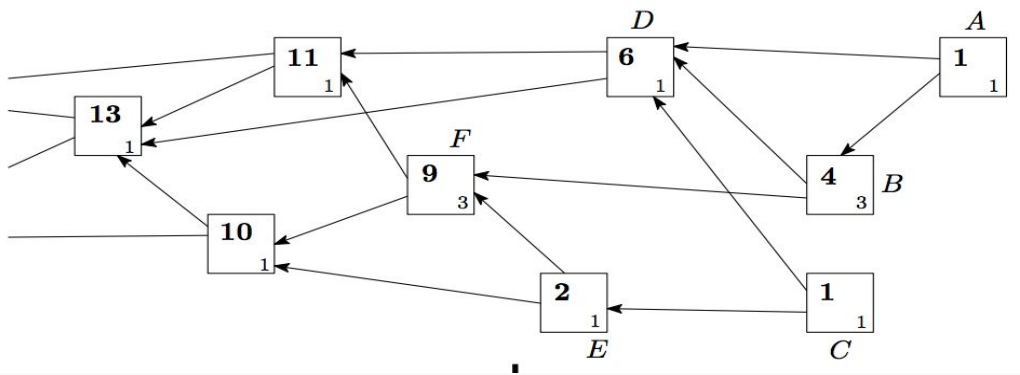
Propagation incentive for nodes

A node will be dropped by its neighbor, when it shows laziness toward propagating transactions. The incentive keeps all nodes working, though they don’t issue transactions that frequent.

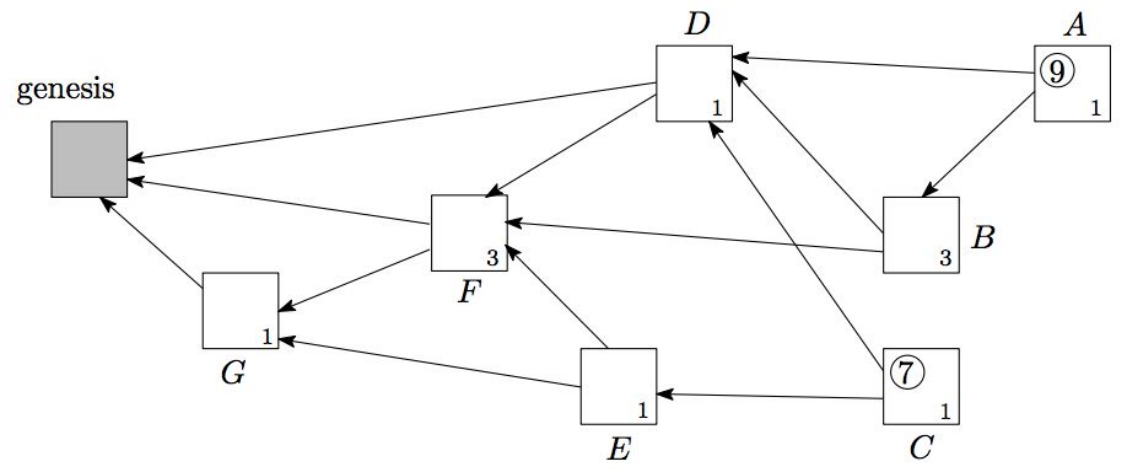
Terms That Matter

Weight (Own weight): The weight of *transaction A* is proportional to the effort put in by its issuer.

Cumulative weight: *Transaction A*'s own weight + the sum of own weights of all the followed transactions that directly/indirectly approve *transaction A*.



Score: *Transaction A*'s own weight + the sum of own weights of all previous transactions approved by *transaction A*



Height: The length of the longest oriented path to the genesis.

Depth: The length of the longest reverse-oriented path to certain tips.

Seeds, Private Keys and Accounts

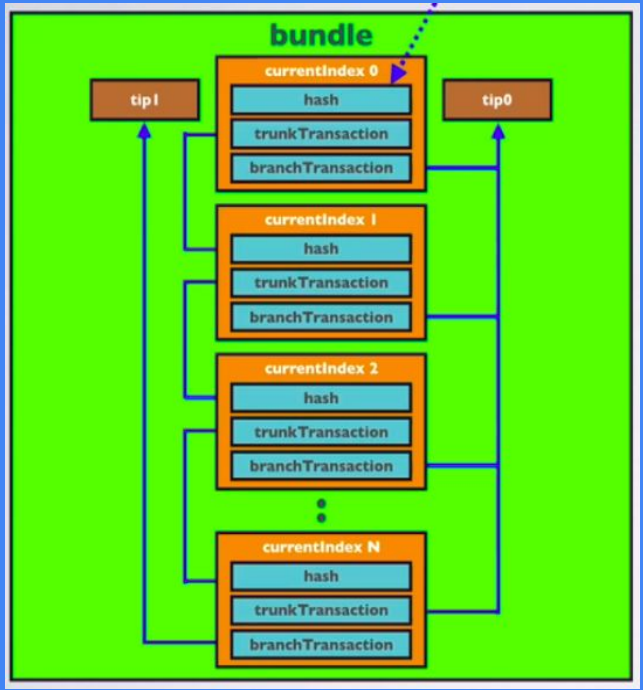
- ➔ In order to create an account with private keys and addresses one need to have a secure seed.
- ➔ It's like a password and username in one.
- ➔ The length can vary from 60-81 characters consisting only A-Z (uppercase) alphabets and the number 9.
- ➔ Generally a seed consists of 81-trytes and is a unique access key to your account and thus your funds. **The seed has to be securely stored.**
- ➔ The seed is only a local stored authentication and is never revealed in a transaction.
- ➔ The private/public key pair is generated deterministically from your **seed + some index + a security level**.
- ➔ From that private key you then generate an address. The key index starting at 0, can be incremented to get a new private key, and thus address.

- We have 3 different security levels which determines the number of rounds for hashing or the length of the private key.

  - Security level 1 : key length 81 trits (security low)
  - Security level 2 : key length 162 trits (security medium)
  - Security level 3 : key length 243 trits (security high)

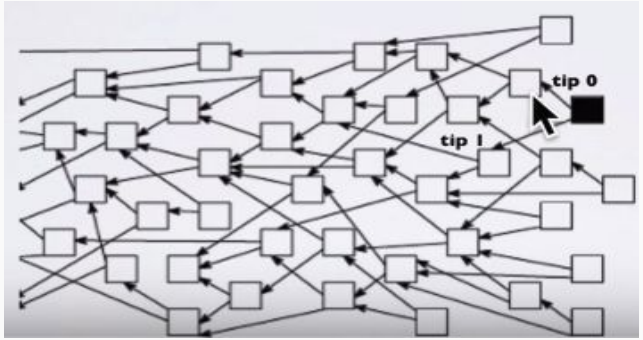
- The security level affects the following two things:
  - How hard it is to brute force a signature for a key.
  - How big the actual signatures are.

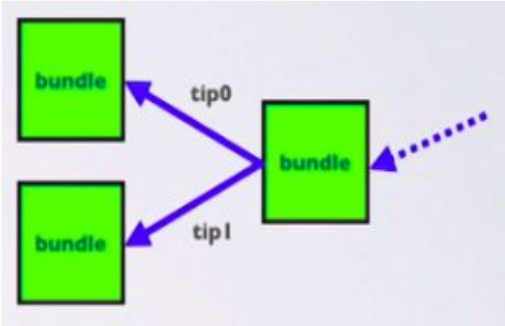
Bundles



currentIndex 0 = tail transaction
currentIndex N = head transaction

Each transaction consists of a bundle of transactions.





Transactions In Bundle

- All transactions in the same bundle should be treated as atomic unit. It means that either all transactions of a bundle are confirmed, or none of them are confirmed.
- Every transaction in the bundle requires it own PoW, hence there are different nonces in the transactions.

Transaction Object

Key Name	Type	Tryte size	Description
Hash	String	81	Uniquely identify the transaction on the tangle.
signatureMessage Fragment	String	2187	Holds either a signature or a message, which may be fragmented over multiple transactions in a bundle.
address	String	81	The address associated with this transaction.
value	int	-	The number of IOTAs being transferred in this transaction.
obsoleteTag	String	27	User-defined tag.
timestamp	int	-	UNIX timestamp when the transaction was issued.
currentIndex	int	-	The transaction’s position in the bundle.

Transaction Object

Key Name	Type	Tryte size	Description
lastIndex	int	-	The last transaction position in the bundle.
bundle	String	81	The bundle hash identifies which transactions belongs to the same bundle.
trunkTransaction	String	81	Tip 0 hash or next transaction hash.
branchTransaction	String	81	Tip 0 hash or Tip 1 hash.
tag	String	27	User defined tag to easily find a transaction.

Transaction Object

Key Name	Type	Tryte size	Description
attachmentTimestamp	int	-	Timestamp after proof of work.
attachmentTimestamp LowerBound	int	-	Attachment time (lower bound) It is a slot for future use.
attachmentTimestamp UpperBound	int	-	Attachment time (upper bound) It is a slot for future use.
nonce	String	27	The proof of work solution.
persistence	bool	-	Indicates if the transaction is pending or confirmed.

Types of Transactions

- **Output Transactions**: Transaction value greater than 0 and address does not belong to the sender.
- **Input Transactions**: In such transaction address belongs to the sender and value can be either positive(non zero) or negative.
- **Meta Transactions**: Zero valued transactions are meta transactions. The signatureMessageFragment of these transactions could either hold a signature or a message fragment.

Types of Input Transactions

- **Input Transactions where the value is negative**: These are the transactions where the complete balance from that address is spent.
- **Input Transactions where the value is greater than zero**: These are the transactions where unspent/not used IOTA’s are send to a new change address in the senders wallet.

Transactions

Transactions is a 3 step process:

1. Constructing the bundles and signing the inputs with private keys.
2. Tip Selection.
3. Proof of Work (PoW)

How IOTA making a transaction ?

Person A has a seed **A_SECRET_SEED** that contain 100i in 4 different address relate to this seed:

seed: A_SECRET_SEED
address[0]: AAAAAA……AAA, balance: 10
address[1]: BBBBBB……BBB, balance: 5
address[2]: CCCCCC……CCC, balance: 25
address[3]: DDDDD……DDD, balance: 60
address[3]: EEEEE……EEE, balance: 0

Person B has a seed **B_SECRET_SEED** that contain 0i in its address:

seed: B_SECRET_SEED
address[0]: QQQQQQ……QQQ, balance: 0

Consider the scenario where person A has to send **80i** to person B **address[0]**.

1. Making transaction bundle

For our scenario, first we need to prepare output transaction, which mean, we want to send to B's address with 80i IOTA:

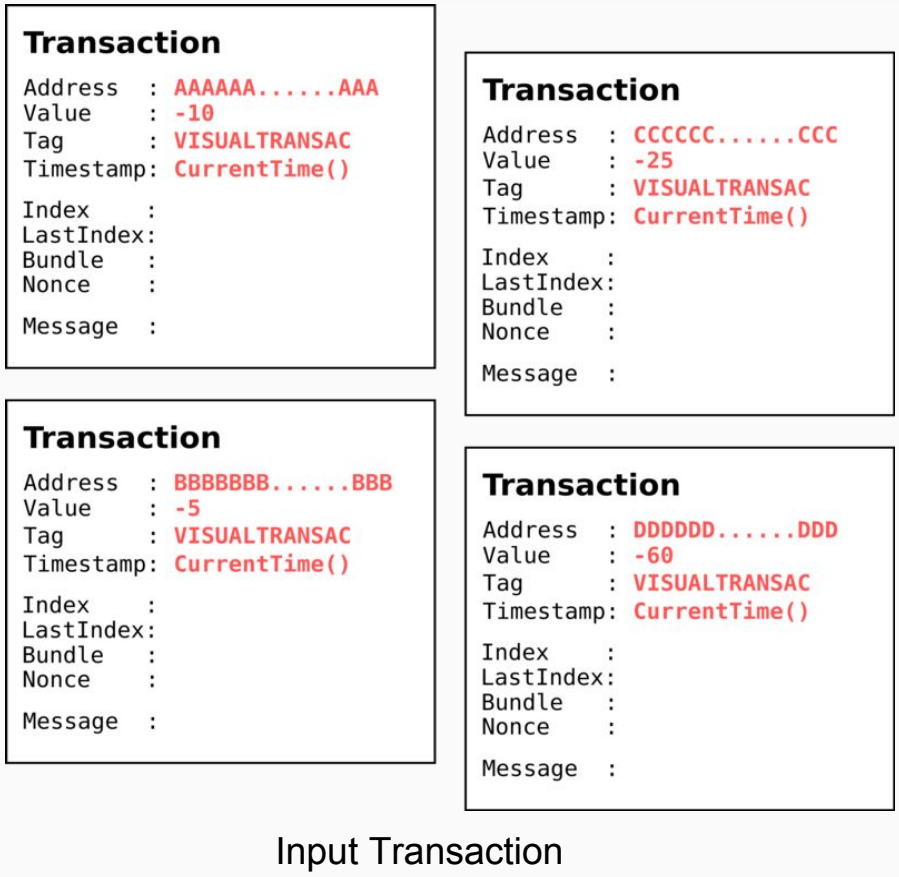
Transaction

```
Address   : QQQQQQ......QQQ
Value     : 80
Tag       : VISUALTRANSAC
Timestamp : CurrentTime()

Index     :
LastIndex :
Bundle    :
Nonce     :

Message   : WELCOME9TO9IOTA
```

Output Transaction

1. Making transaction bundle

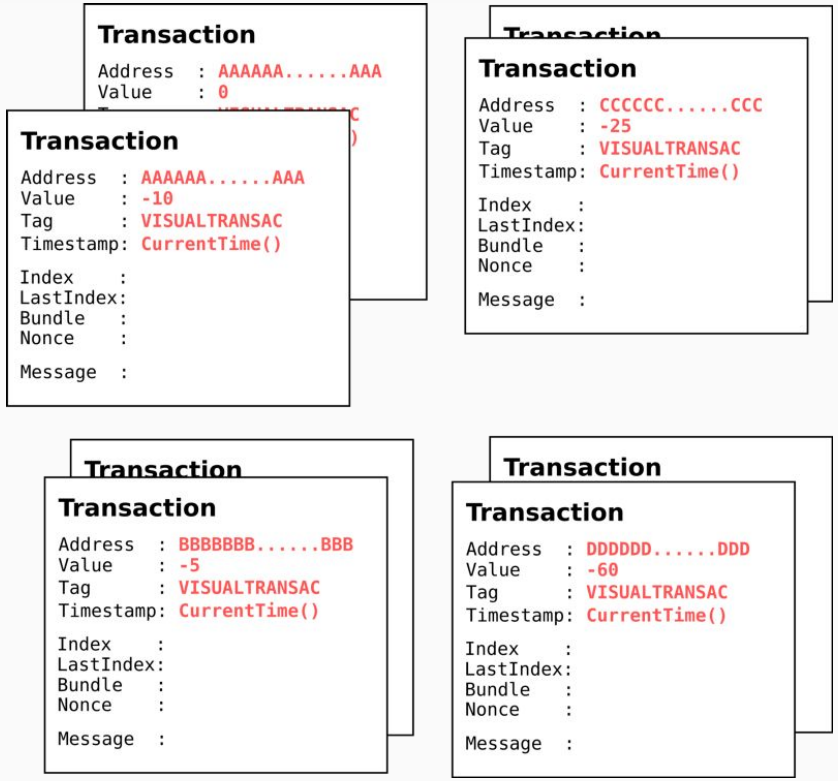
Next, we will need to prepare input transaction. In our scenario, we will need to use all four address that contain IOTA (10 + 5 + 25 + 60 > 80) to fulfill output value 80i.



Input Transaction

1. Making transaction bundle

But our input transaction need to contain transaction signature, default address security level is 2, that mean we need an additional meta transaction to carry the transaction signature.



Input Transaction with meta transactions

1. Making transaction bundle

Now we have an unbalanced bundle.
Take a little count, we have 10 + 5 + 25 + 60 = 100 IOTA input, and 80 IOTA output, which mean this bundle still got 100 - 80 = 20 IOTA unspent.
We will need to get an additional transaction to receive this unspent IOTA.

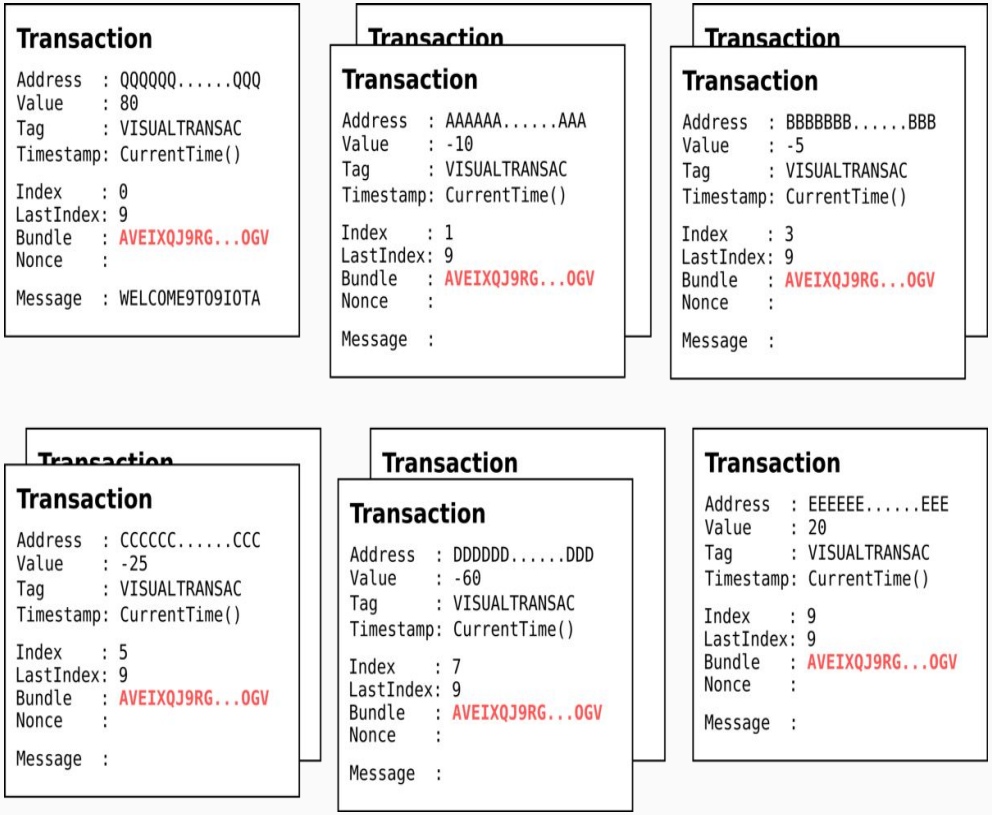
Transaction

```
Address   : EEEEEE.....EEE
Value     : 20
Tag       : VISUALTRANSAC
Timestamp: CurrentTime()

Index     :
LastIndex:
Bundle    :
Nonce     :

Message   :
```

Unspent Transaction

2. Finalize bundle

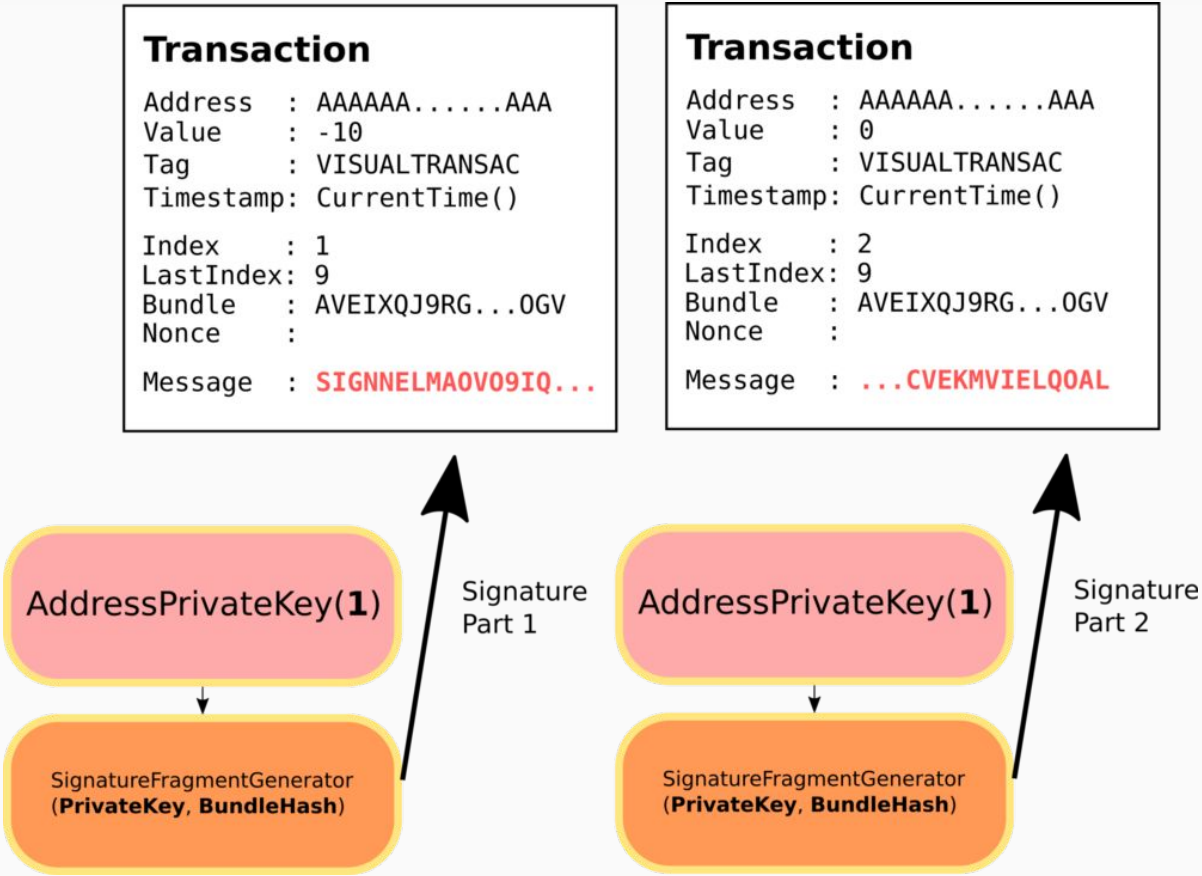
In this step, we will fill in transaction index, last index and generate bundle hash by Kerl hash function.



3. Signing signature for input transactions

- In IOTA: the private/public key pair is generated deterministically from your seed + some index (+ a security level).
- we will need to sign the input transactions with correspond address’ private key.
- Get address private key via key generate with address index and security level.
- Using address private key and bundle hash to generate signature fragment and filling into transaction’s signature fragment part.
- If the security level is 2, then we will need to sign up two transaction (1 for input transaction and the following meta transaction).
- At the end of this step, we will get a list of transaction trytes that including bundle hash and transaction signature.

3. Signing signature for input transactions



4. Getting two tips—trunk and branch

Next, we'll select TWO tips based on the algorithms that will be discussed later in the presentation and the corresponding trunk and branch values will be filled.

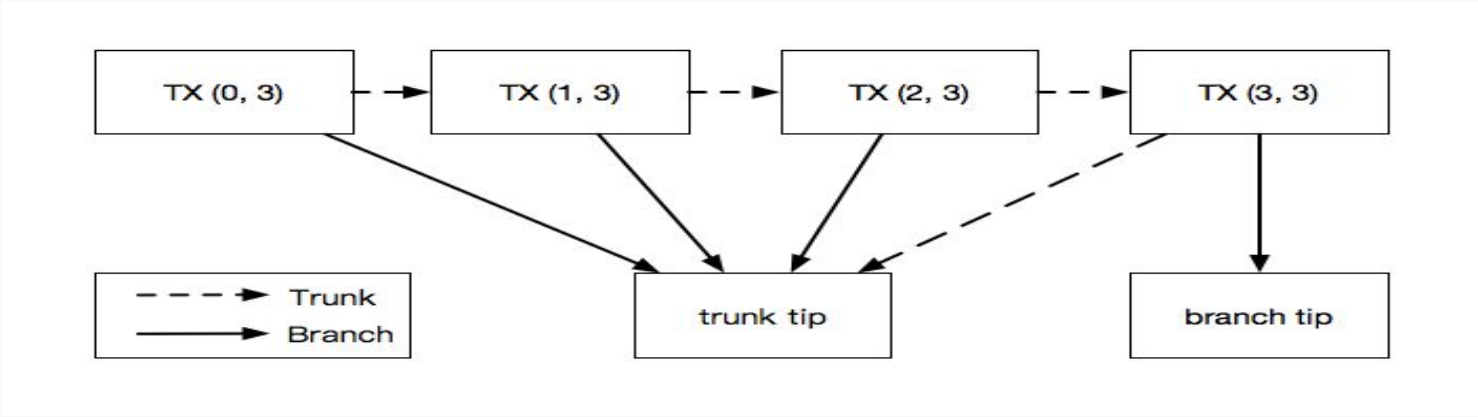
5. Proof of Work

- IOTA uses PoW for spam protection, similar in spirit to the PoW used in Hashcash. This is a short computational operation.

- **Min Weight Magnitude:**
  This refers to the number of trailing zeros (in trits) in transaction hash.
  The device which does the PoW will bruteforce the transaction hash to find a nonce that has the correct number of trailing 0's.

5. Proof of Work

- We will need to fill-up trunk, branch, and find nonce (Proof of Work) into each transaction in the bundle.
- It will then walk through all transactions in the bundle from the last index to 0 index, to fill-up everything that is stated above.
- Last index's transaction trunk and branch hash will be previous tips we get. Other transaction's trunk will be previous transaction's hash, and branch hash is trunk transaction from tips.



Tip Selection

We call unapproved transactions tips.

Each incoming transaction needs to choose two tips to approve.

The strategy for choosing which two tips to approve is very important, and is the key to IOTA’s unique technology.

Tip selection

Random Selection of Tip:

Each incoming transaction looks at all the currently unapproved transactions, and simply chooses two at random.

Visual Simulation

Tip Selection

Unweighted Random Walk:

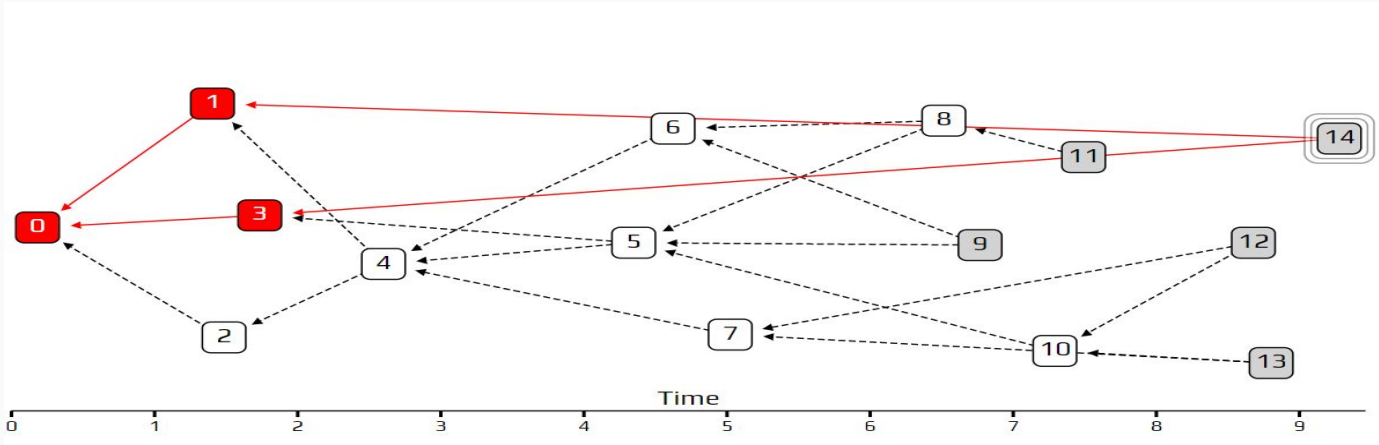
- Using this algorithm, we put a walker on the genesis transaction, and have it start “walking” towards the tips.
- On each step it jumps to one of the transactions which directly approves the one we are currently on.
- We choose which transaction to jump to with equal probability.

Visual Simulation

Tip Selection

Lazy Tips:

- A lazy tip is one that approves old transactions rather than recent ones.
- It only broadcasts its own transactions based on old data.
- This does not help the network, since no new transactions are confirmed.



Tip Selection

Lazy Tip continued...

- If we use the uniform random tip selection algorithm, transaction 14 is just as likely to get approved as any other, so it is not being penalized at all.
- Using the unweighted walk, this bad behavior is even encouraged.

Tip Selection:

Can we force participants to only approve recent transactions ?

No.

This would clash with the whole idea of decentralisation. Transactions can approve whomever they please.

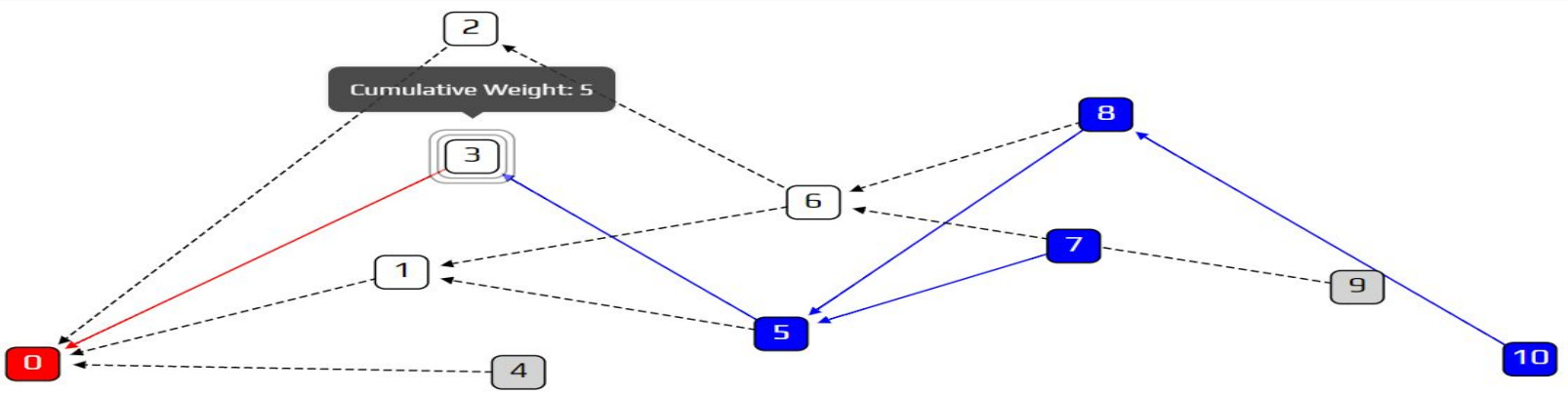
One solution is to construct a system with built-in incentives against such behavior, so that lazy tips will be unlikely to get approved by anyone.

Tip Selection

The weighted random walk:

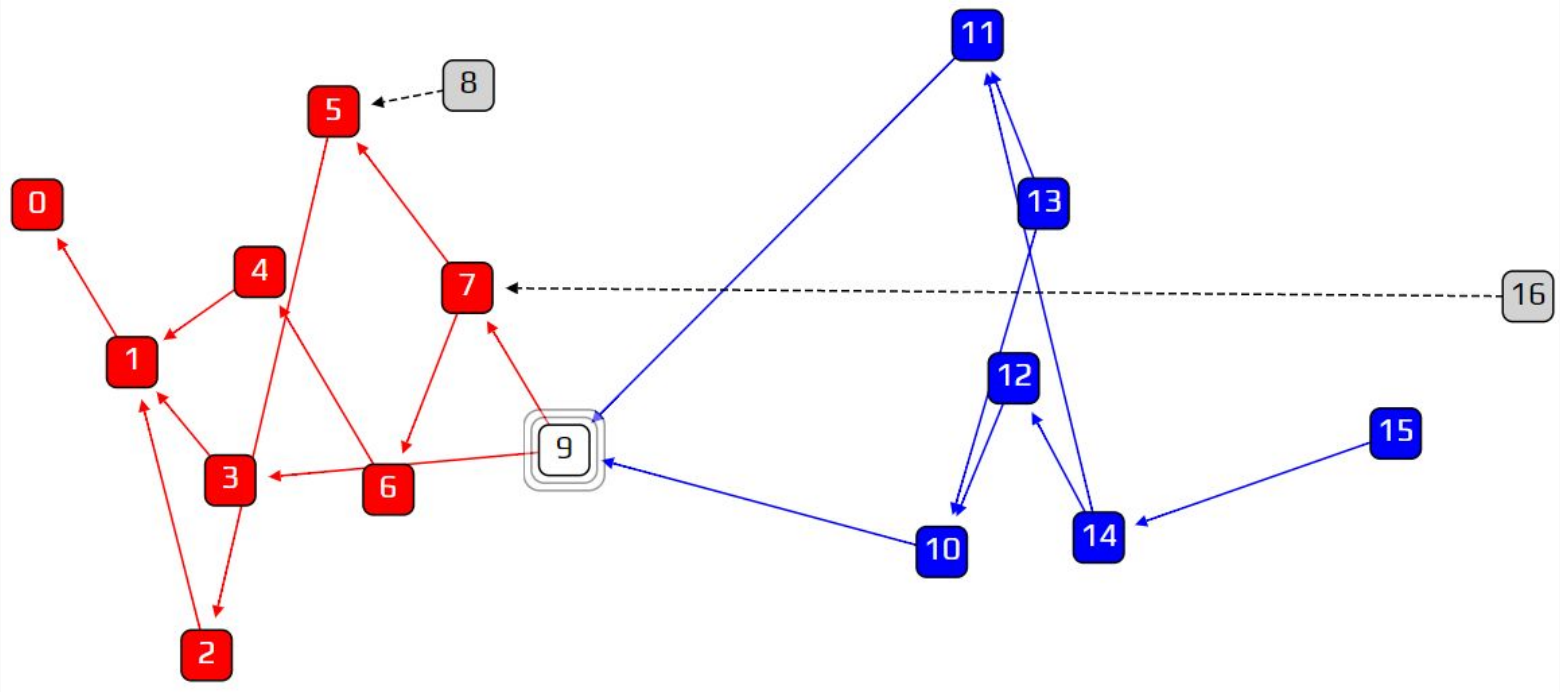
Our strategy will be to bias our random walk, so we are less likely to choose lazy tips.

Cumulative weight: we count how many approvers a transaction has, and add one. We count both direct and indirect approvers.



Tip Selection

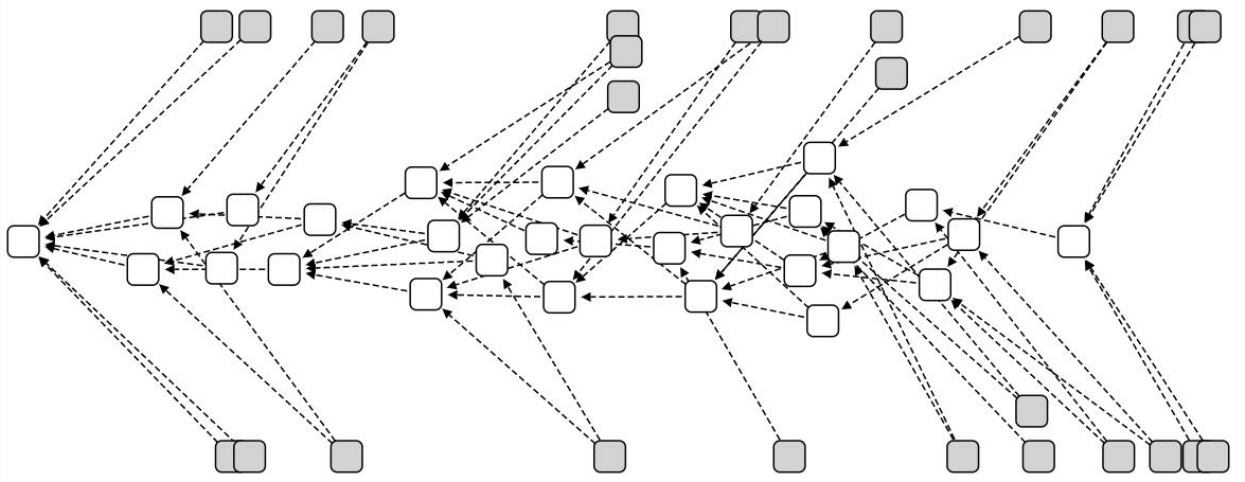
This is an effective way of discouraging lazy behavior.



Tip Selection

Do we need randomness at all ?

Super Weighted Random Walk: each junction we choose the heaviest transaction, with no probabilities involved.



Tip Selection

We need a parameter to control how strong our walk is.
This introduces our new parameter α, which sets how important a transaction’s cumulative weight is.

This method of setting a rule for deciding the probability of each step in a random walk is called a **Markov Chain Monte Carlo technique**, or **MCMC.**

Confirmation Confidence

This confidence is a measure of a transaction’s level of acceptance by the rest of the tangle.

The confirmation confidence of a transaction is computed as follows:

1. Run the tip selection algorithm 100 times.
2. Count how many of those 100 tips approve our transaction, and call it A.
3. The confirmation confidence of our transaction is “A percent”.

Visual Simulation

The Coordinator

- To make it possible for the network to grow and protect it against certain attacks, IOTA currently relies on a coordinator.
- The coordinator checkpoints valid transactions, which are then validated by the entire network.
- The coordinator issues periodic milestones, which reference valid transactions.
- When the network is mature enough to get rid of the Coordinator the network will also instantly become orders of magnitude more efficient.

Stability

Assumptions :

- All devices have same computing power.
- At any point when a node issue a transaction, what it sees is not the actual state of tangle but the one exactly h minutes ago.
- Number of tips should remain roughly stationary in time

Stability

- Number of tips (**L(t))** at any time **t** should be finite.
- $P(\{L(t) - L(t - h)\} = n) = \mathbf{e^{-ƛh}(ƛh)^n/n!}$
- The mean number of chosen tips is **2r/(r + ƛh)**
- **L = 2ƛh**

r - initial revealed tips,
ƛ - rate of transaction,
h - average time a device need to perform calculation to issue a transaction.

Load Regimes

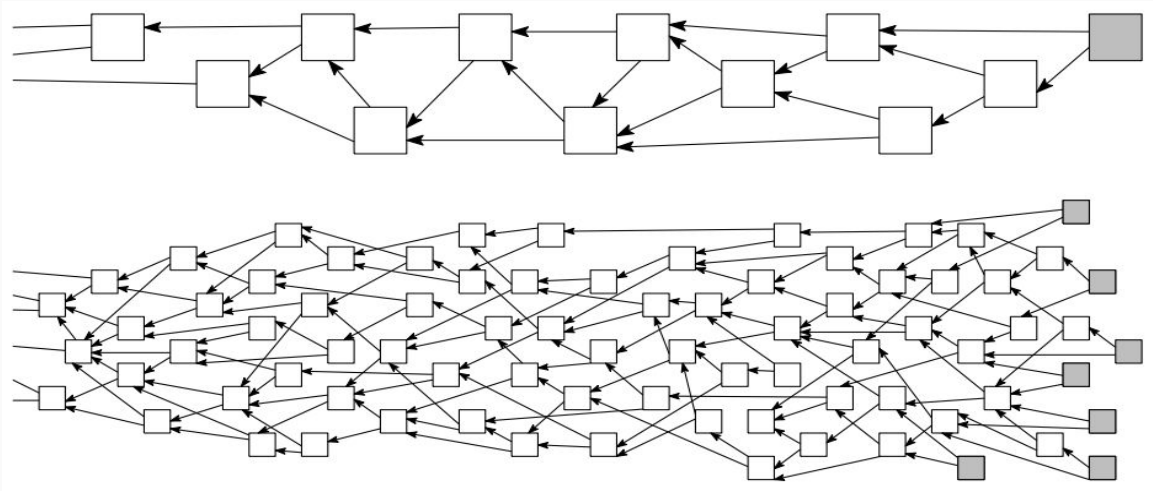
Low Load

- Number of tips are small.
- Flow of Transaction is so small that it is not probable that several transaction approve the same tip.
- **Happened when :** low network latency and great computational power.

High Load

- Number of tips are large.
- A strong inflow of transaction.

- **Happened when :** high network latency and low computational power
- **Disadvantage :** Some transactions might need to wait for a long while before being approved.



Low load (top) and high load (bottom) regimes

Conclusions

- If a transaction uses the strategy of approving two random tips, the typical number of tips is given by **$L = 2\lambda h$**.
- There are only a few tips in the low load regime. A tip gets approved for the first time in **$\Theta(\lambda^{-1})$** time units, where **λ** is the rate of the incoming flow of transactions.
- The typical time for a tip to be approved is **$\Theta(h)$** in the high load regime, where **h** is the average computation/propagation time for a node

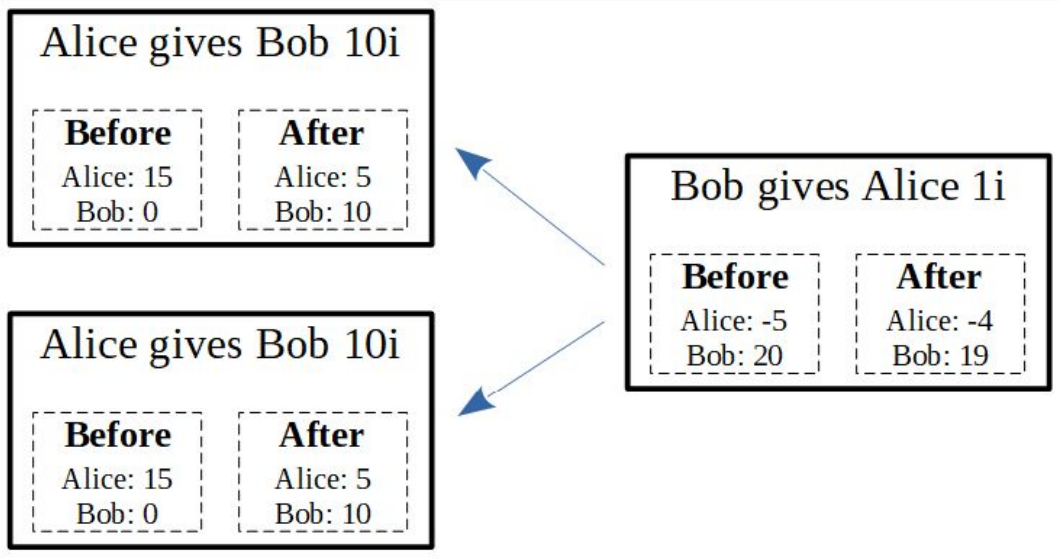
- After a transaction gets approved multiple times in the low load regime, its cumulative weight will grow with speed **λw**, where **w** is the mean weight of a generic transaction.

- In the high load regime, there are two distinct growth phases. First, a transaction's cumulative weight **H(t)** grows with increasing speed during the adaptation period according to : **H(t)** ≅ **exp(**0.352**t/h)**;

- After the adaptation period is over, the cumulative weight grows with speed **λw.**

Possible Attack Scenarios

Double Spending-



Double Spending

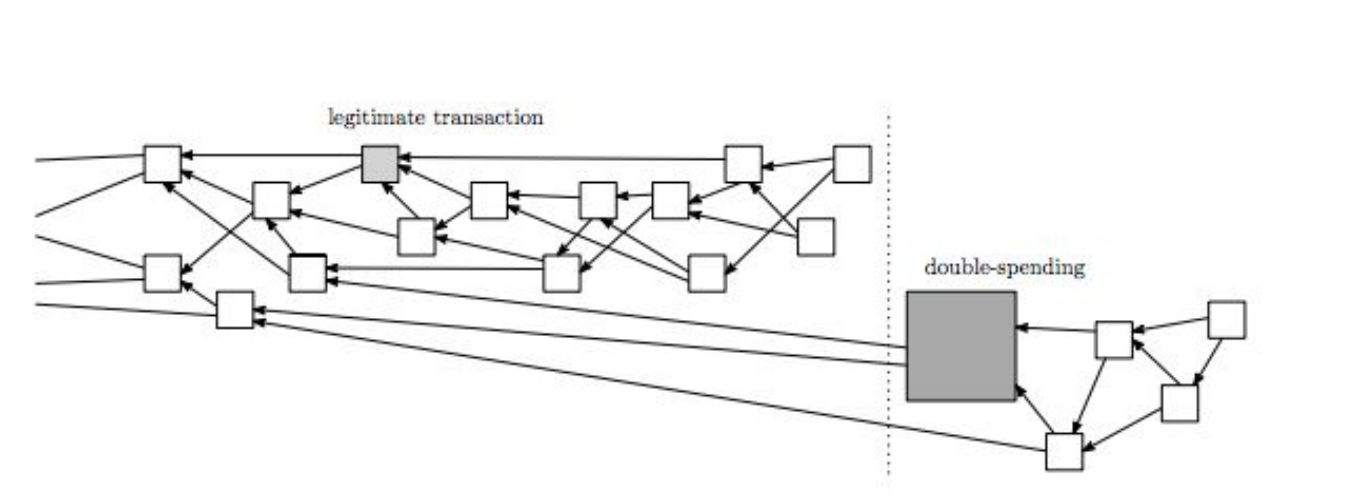
- This last situation is called a **double spending**, because Alice spent her money *twice*. Notice that she did not break the protocol, because she had enough money for each individual transaction.

- She did however create two *branches* in the tangle that cannot be reconciled. This creates a problem for honest users of IOTA: which branch should they approve?

- The solution to this problem is once again the weighted walk. Eventually one of the branches will grow heavier than the other, and the lighter one will be abandoned

- This also implies that a transaction cannot be considered to be confirmed immediately after it is issued, even if it has some approvers, since it might be part of a branch that will be abandoned eventually.

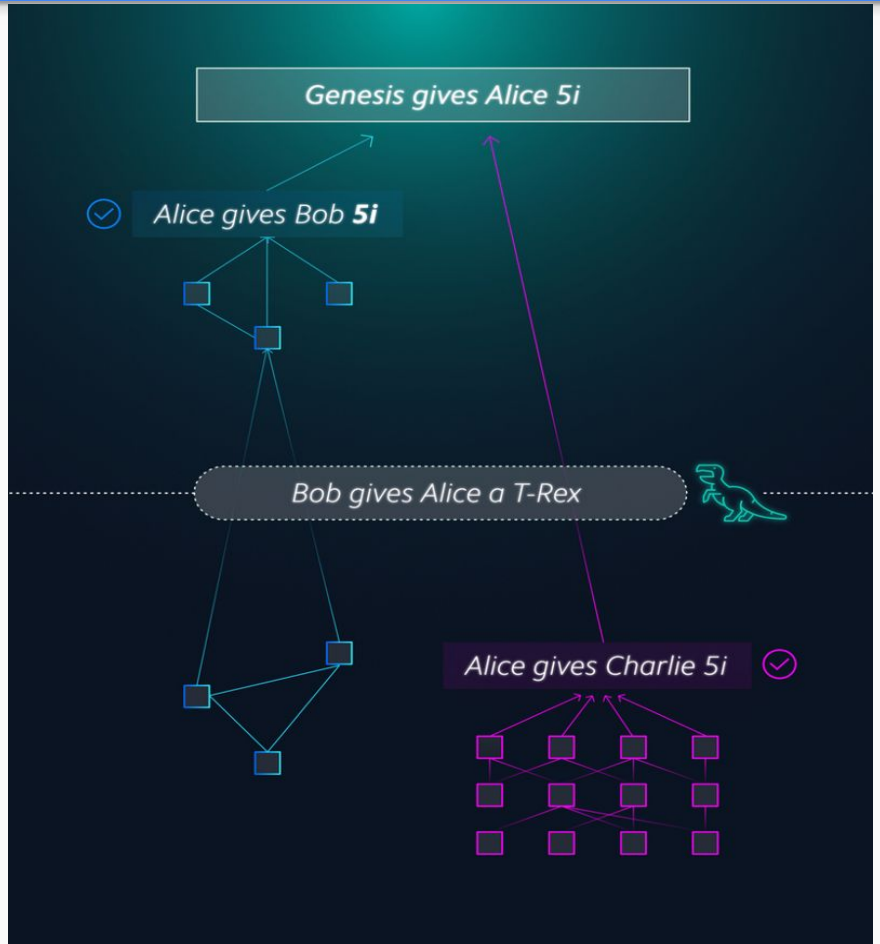
Large Double Spending Attack



Possible Solutions

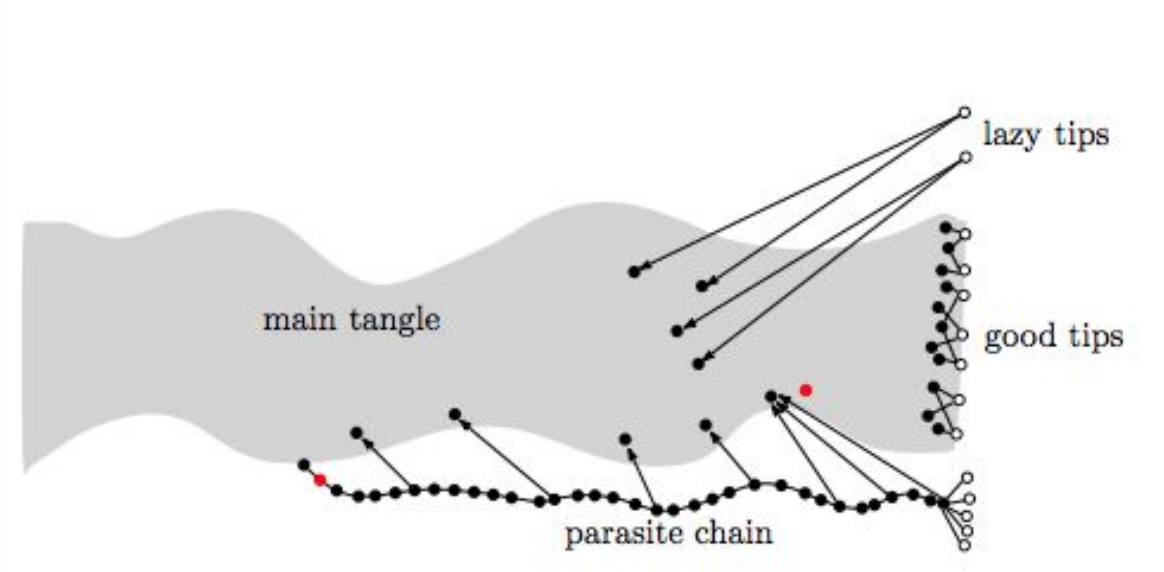
- Limiting the own weight of each node, or even setting it to a constant value.(Not Possible because of spaming)
- The inequality $\lambda > \mu$ should be true for the system to be secure. In other words, the input flow of “honest” transactions should be large compared to the attacker’s computation

Parasite Chain Attack



Parasite Chain Attack Solutions

- **Using N particles while using random walk algorithm (MCMC).**
- **It is easy to see why the MCMC selection algorithm will not select one of the attacker's tips with high probability in the following.**



Parasite Chain Attack Solutions

As an additional protecting measure, we can first ran a random walk with a large α (so that it is in fact “almost deterministic”) to choose a “model tip”; then, use random walks with small α for actual tip selection.

Thanks!

Nitesh Gupta (22)

Anurag Bhardwaj (28)

Ayan Sheikh (29)

Arpit Bahety (34)

Kautish Jaiswal (51)