The IOTA Usecase

Since the financial crisis of 2007, discussions about the blockchain have begun and established a new topic not only in fintech but also in industry and other fields.

We are used nowadays to talk about Blockchain having in our mind cryptocurrencies (especially Bitcoin), but the actual needs must focus us to more urgent matters.

In a distributed ledger, the data is always stored in blocks and the trust is assured by the decentralization structure and the embedded security. For Internet of Things, the security and computational power can be issues in implementing blockchain technology. But, if we integrate the data of IoT devices in a blockchain infrastructure we can see a lot of possibilities related to the usability in the real world: for voting, for trusting data (like temperature, pollution parameters, source of provenience, product tracking) so on.

The usecase that I want to talk about is IOTA.

What is IOTA

Long story short (their motto): *IOTA's Tangle is an open, feeless and scalable distributed ledger, designed to support frictionless data and value transfer*.

As a distributed ledger technology, provides a trust layer for any devices that are connected to the global internet.

Through its open network of nodes, allows you and your devices to:

- Use the network as a source of truth for stored data
- Transfer value in IOTA tokens

IOTA networks are peer-to-peer networks where no central authority controls the data and all nodes hold a copy of it reaching a consensus on its contents.

IOTA is for anyone who:

- does no trust centralized networks
- wants to secure their data
- values security
- wants the freedom to transact

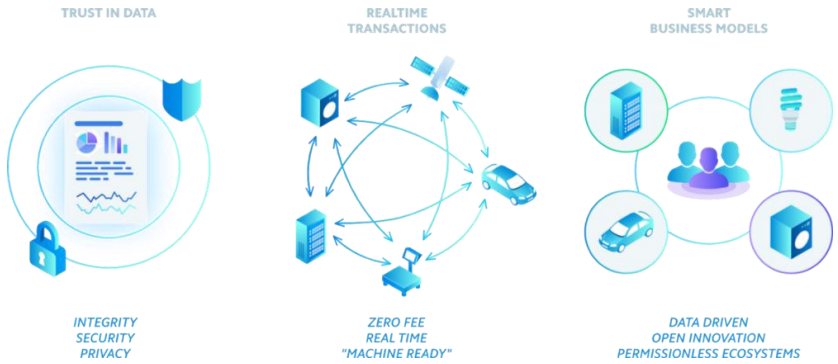

Figure 1. Who is IOTA for (Source: iots.org)

History of IOTA

IOTA was created in 2015 by David Sønstebø, Dominik Schiener, Sergey Ivancheglo and Serguei Popov. Initial development was funded by an online public crowdsale, with the participants buying the IOTA value token with other digital currencies. Approximately 1300 BTC were raised, corresponding to approximately 500,000 USD at that time and the total token supply was distributed pro-rata over the initial investors. The network went live in 2016.

Architecture of IOTA

Figure 2. Architecture of IOTA (source: iota.org)

The Tangle: a public ledger (based on a *Directed Acyclic Graph*) that is replicated across all nodes in an IOTA network. All data in the Tangle is stored in objects (transactions).

As a DAG (aka Directed Acyclic Graph), it is possible to find shortest or longest paths from a given starting vertex in linear time by processing the vertices in a topological order and calculating the path length for each vertex to be the minimum or the maximum length obtained via any of its incoming edges.

Nodes: interconnected devices that are responsible for ensuring the integrity of the Tangle

Clients: users of an IOTA network who send transactions to nodes to be attached to the Tangle.

Manual peering

In order to join the Tangle, a node is required to connect to some existing nodes (peering). The current IRI software only permits manual peering, i.e., a node operator has to manually look for the addresses of other Tangle's nodes. Peering is fundamental to propagate transactions and to synchronize to the current status of the ledger. As for the latter, milestones are useful anchors to determine whether two nodes have fallen out of synchronization: If a node's latest solid milestone is much older than its peers', it is probably lagging behind.

Rate control mechanism

In order to issue a transaction, a node must solve a cryptographic puzzle (Proof-of-Work). This is necessary to guarantee that nodes do not arbitrarily spam the network, or to avoid that they inject more transactions than the network can handle.

Tip selection strategy

Approving transactions is a fundamental procedure which leads to the DAG structure of the Tangle. To approve a transaction, a node must verify that no inconsistencies with respect to the ledger state are introduced. Although it is not possible to enforce which transaction to validate, the original IOTA white paper suggests a tip selection algorithm based on a random walk which:

- discourages lazy behavior and encourages approving fresh tips;
- continuously merges small branches into a single large branch, thus increasing confirmation rate;
- in case of conflicts, kills off all but one of the conflicting branches.

Consensus

The main role of milestones is to determine the consensus.

The Tangle applies a simple rule: A transaction is confirmed if and only if it is referenced by a milestone. In IRI, this is reflected in the getBalances and getInclusionStates API calls, which indicate how many tokens an account has and whether a transaction is confirmed, respectively.

IOTA reference code optimization

Instead of computing the full ledger state starting from the genesis, an intermediate state is saved for each milestone; similarly, milestones are used in local snapshots, i.e., the IRI pruning mechanism, which allows nodes to avoid storing older parts of the Tangle.

Balanced ternary system

Perhaps the prettiest number system of all, is the balanced ternary notation.

Donald E. Knuth, The Art of Computer Programming

All transactions in the Tangle are made up of characters called trytes. IOTA uses the ternary numeral system to represent data because, compared to binary, ternary computing is considered to be more efficient as it can represent data in three states rather then just two.

In IOTA, data is represented in balanced ternary, which consists of 1, 0 or -1 (called trits). Three of them are equal to one tryte, which can have 3^3=27 possible values.

Tryte encoding

To make trytes easier to read, they are represented as one of 27 possible tryte-encoded characters, which consist of the number 9 and the uppercase letters A-Z (indexed from 0 to 26).

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
9	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	**N**	**O**	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**

Example of encoding data.

Let's suppose that we want to encode the string "**Hello word!**". Each ASCII character from the given string will be replaced by a pair of trite characters.

We have the following conversions:

ASCII Character	**H**	**e**	**l**	**l**	**o**	**Space**	**w**	**o**	**r**	**l**	**d**	**!**
Decimal value	72	101	108	108	111	32	119	111	114	108	100	33
Tryte pair	RB	TC	9D	9D	CD	EA	KD	CD	FD	9D	SC	FA

The conversions are started from the decimal value of each ASCII character:

- First, we compute the remainder of the division at 27 (for H, we will obtain 72%27=18) and this will be the index of first tryte character (for H, will be R);
- Second, we compute the quotient of the division at 27 (for H, we will obtain (72-18)/27=2) and this will be the index of second tryte character (for H, will be B).

For the given string, we will obtain:

"RBTC9D9DCDEAKDCDFD9DSCFA"

Sending data to the Tangle (2019 version)

First, we must setup the environment using the command line (for Windows OS) or terminal (for MacOS or Linux OS):

- We create a folder (for example, iota-app) for our application

mkdir iota-app

cd iota-app

- We initialize the Node.js project:

npm init

- We install the IOTA core client libraries:

npm install @iota/core @iota/converter

- We run the script.js file (below) from node and we will obtain a hash string (like KJQLEPCOIOHIQEZMO9MLHZEIAWHGZZSFJBRREKMI9XZGJJDIQFVZPRCRZTKJHFVCBEIOIULARRRMHO999):

node script.js

where the file script.js is below:

```
// script.js

const Iota = require('@iota/core');

const Converter = require('@iota/converter');

// Connect to a node

const iota = Iota.composeAPI({

  provider: 'https://nodes.devnet.iota.org:443'

});



const depth = 3;

const minimumWeightMagnitude = 9;
```

```
// Define a seed and an address.

// These do not need to belong to anyone or have IOTA tokens.

// They must only contain a mamximum of 81 trytes

// or 90 trytes with a valid checksum

const address =
'HEQLOWORLDHELLOWORLDHELLOWORLDHELLOWORLDHELLOWORLDHELLOWORLDHELLOWORLD
HELLOWOR99D';

const seed =
'PUEOTSEITFEVEWCWBTSIZM9NKRGJEIMXTULBACGFRQK9IMGICLBKW9TTEVSDQMGWKBXPVCBMM
CXWMNPDX';


// Define a message to send.

// This message must include only ASCII characters.

const message = JSON.stringify({"message": "Hello, world! Today is June 15, 2021, and we are at
Blocks Summer School!"});


// Convert the message to trytes

const messageInTrytes = Converter.asciiToTrytes(message);


// Define a zero-value transaction object

// that sends the message to the address

const transfers = [

 {

   value: 0,

   address: address,
```

```
    message: messageInTrytes
  }
];

// Create a bundle from the `transfers` array
// and send the transaction to the node
iota
  .prepareTransfers(seed, transfers)
  .then(trytes => {
    return iota.sendTrytes(trytes, depth, minimumWeightMagnitude);
  })
  .then(bundle => {
    console.log(bundle[0].hash);
  })
  .catch(err => {
    console.error(err)
  });
```

- We can find, using the hash string, the message that we have sent in the https://explorer.iota.org/legacy-devnet:

https://explorer.iota.org/legacy-devnet/transaction/**message-response-hash**

New features and components

Starting with the second major version of IOTA, we have more powerful tools for the architecture of this distributed ledger.

Autopeering

In the IOTA protocol, a node (or peer) is a machine storing the information about the Tangle, IOTA's underlying data structure. In order for the network to work efficiently and for the nodes to be kept up-to-date about the ledger state, nodes exchange information, such as new transactions, with each other.

Each node establishes a communication channel with a small subset of nodes (i.e., neighbors) via a process called peering. The peering process is potentially vulnerable to various attacks. For instance, if all of a node's neighbors are controlled by an attacker, then the attacker has complete control over the node's view of the Tangle, leaving the victim node vulnerable to a host of scams. This type of attack is known as an Eclipse attack.

Currently, in the mainnet IOTA network, nodes use a manual peering process to mutually register as neighbors. However, manual peering might be susceptible to these attacks (including social engineering ones) and can be also generally inconvenient. To this end, and to simplify the setup process of new nodes, we introduce a mechanism that allows nodes to choose their neighbors automatically. The process of nodes choosing their neighbors without manual intervention by the node operator is called *autopeering*.

Specifically, in this section we propose an autopeering mechanism which achieves two important goals: First, it creates an infrastructure where new nodes can easily join the network; second, we make sure that an attacker cannot target specific nodes during the peering process, i.e., we ensure the network to be secure against Eclipse attacks.

For more technical details, we refer the interested reader to the source code of our autopeering simulator as well as its integration on *GoShimmer*.

Mana system

The principles of a mana system is that one should get or have more mana the more one contributes to the network. Contribution is naturally associated to how much stake one holds, but although having tokens helps the network, one should not be able to "mine" an unrestrained amount of mana by simply holding some quantity of tokens for a large amount of time, or by frequently sending tokens around.

Now we will define a mana system that has the described properties and conclude the subsection with a possible improvement that would keep abrupt variations in mana from happening.

To achieve this we introduce three new concepts:

- Pending mana. Addresses generate pending mana at a rate proportional to the stake they hold.
- Mana. When funds (i.e., IOTA tokens) are spent from an address, the pending mana that has been generated by this address, is converted to mana and pledged to a node.
- Pending mana is now generated by the funds on the receiver's address.
- Decay. Both mana and pending mana decay at a rate proportional to its value, hence keeping mana from growing unrestrained over time.

This model has two desired properties:

1. Mana cannot be gamed, in the sense that the amount of mana received will not change depending on how often the user converts pending mana into mana or in how many addresses the tokens are split.
2. Mana is bounded, both for a node with addresses summing tokens and, therefore, also for the entire network.

This approach has some interesting features:

- Once an address gains control over the tokens, they must wait for their pending mana to gain their full potential.
- Mana is easy to store and is snapshotable: its decay and growth can be deduced from changes in time and balance: if a nodes' pending mana and balance at t1 is known, then its potential mana at time t2 can be deduced for any t2 > t1.

This proposal of mana satisfies most of our needs but it still could be criticized for its abrupt variations over time: since we could have a large quantity of pending mana becoming mana for a node at once, many applications that depend on how much mana a node currently has could abruptly change their vision.

Fast probabilistic consensus

Increasingly, distributed systems need to provide a consensus on the current state of the infrastructure within given time limits, and to a high degree of accuracy. At the core of cryptocurrency transactions, for example, is that miners must achieve a consensus on the current state of transactions. This works well when all the nodes are behaving correctly, but a malicious agent could infect the infrastructure, and try and change the consensus.

Suppose that there is a network composed of *n* nodes, and these nodes need to come to consensus on the value of a bit. Some of these nodes, however, may belong to an adversary, an entity which aims to delay the consensus or prevent it from happening altogether. This use case focuses on

this situation - and which is typical in the cryptocurrency applications - when the number n of nodes is large, and where they are possibly (geographically) spread out. This makes the communicational costs important whereas computational complexity and the memory usage are often of a lesser concern.

Let us now explain informally what makes this protocol converge fast to the consensus even in the Byzantine setting. The general idea is the following: if the adversary (Eve) knows the decision rules that the honest nodes use, she can then predict their behaviour and adjust her strategy accordingly, in order to be able to delay the consensus and further mess with the system. Therefore, let us make these rules unknown to all the participants, including Eve. Specifically, even though Eve's nodes can control (to some extent) the expected proportion of 1-responses among the *k* queries, she cannot control the value that the threshold random variable assumes. As a consequence, the decision threshold X1 will likely be "separated" from that typical proportion.

When this separation happens, the opinions of the honest nodes would tend very strongly in one of the directions whp. Then, it will be extremely unlikely that the system leaves this "pre-consensus" state, due to the fact that the decision thresholds, however random, are always uniformly away from 0 and 1. Also, we mention that a similar protocol with intended cryptocurrency applications was considered in the article "Scalable and probabilistic leaderless bft consensus through metastability" (by T. Rocket, M. Yin, K. Sekniqi, R. van Renesse, E.G. Sirer, 2019). However, there only "fixed thresholds" were used, which gives Eve much more control, so that, in particular, then she could delay the consensus a great deal. As a last remark, it is important to note that having "independently random thresholds" (i.e., each node independently chooses its own decision threshold) is not enough to achieve the effect described above - these "locally random" decisions will simply average out; that is, having common random numbers is indeed essential.

Chrysalis

The first IOTA network is centralized, a transaction on the network is considered valid if and only if it is referenced by a milestone issued by a node operated by the IOTA foundation called the coordinator. In 2019 the IOTA Foundation announced that it would like to operate the network without a coordinator in the future, using a two-stage network update, termed Chrysalis for IOTA 1.5 and Coordicide for IOTA 2.0. The Chrysalis update went live on 28 April 2021 and removed its controversial design choices such as ternary encoding and Winternitz one-time signatures, to create an enterprise-ready blockchain solution. In parallel, Coordicide, which it will be the next network, is currently developed, to create a distributed network that no longer relies on the coordinator for

consensus. A testnet of Coordicide was deployed late 2020, with the aim of releasing a final version in 2021.

Hornet

HORNET is a powerful, community driven IOTA node software written in Go. It is easy to install and runs on low-end devices like the Raspberry Pi 4. a group of community developers alongside the IOTA Foundation are responsible for building and maintaining Hornet. Hornet is a full-fledged software providing full node capabilities including full support of the Chrysalis network update.

By running your own node you have the following benefits:

- You have direct access to an IOTA network, instead of having to connect to and trust someone else's node.
- You help the IOTA network to become more distributed and resilient by validating messages and value transactions in the IOTA network,

The source code of the project is available on https://github.com/gohornet/hornet.

More informations: https://hornet.docs.iota.org/.

Bee

The IOTA Foundation aims to allow machines of all performance levels to contribute to the IOTA network, from microcontrollers to phones, web browsers, and servers.

Therefore, we are developing Bee as a modular collection of extendable crates, which expose foreign function interfaces (FFIs) for the next iteration of client libraries.

Bee will be a central reference implementation for the most important data structures and algorithms. This implementation will be verified during a Request for Comments (RFC) process, and eventually certified.

By using this approach, we hope that improvements to core components will quickly propagate to all other client libraries, rather than having to fix each one individually.

More informations: https://bee.docs.iota.org/.

Stronghold

Stronghold is an open-source software library that was originally built to protect IOTA Seeds, but can be used to protect any digital secret.

It is a secure database for working with cryptography, which ensures that secrets (like private keys) are never revealed - but can be used according to best practices.

It provides its own peer-to-peer communication layer, so that different apps can securely communicate using the state-of-the-art Noise Protocol over libp2p.

IOTA Stronghold is a secure software implementation with the sole purpose of isolating digital secrets from exposure to hackers and accidental leaks. It uses encrypted snapshots that can be easily backed up and securely shared between devices. Written in stable rust, it has strong guarantees of memory safety and process integrity.

There are four main components of Stronghold:

- Client: The high-level interface to Stronghold (prefers Riker, functional integration also available)
- Engine: Combines a persistence store (Snapshot) with an in-memory state interface (Vault) and a key:value read/write (Store).
- Runtime: Is a process fork with limited permissions within which cryptographic operations take place.
- Communication: Enables Strongholds in different processes or on different devices to communicate with each other securely.

More informations: https://stronghold.docs.iota.org/

Wallet

Trinity is a mobile and desktop application with a user interface that allows you to transfer data and IOTA tokens. With Trinity you can encrypt and store one or more seeds, where each one has its own account, transaction history, and settings.

Trinity wallet allows you to do the following:

- Create a password-protected account to store and access your seeds
- Read your balance and transaction history
- Send and receive transactions

To use Trinity, you must enter your seed. Therefore, to ensure the security of your IOTA tokens, Trinity has been audited by external parties.

Trinity was replaced be Firefly wallet on April 28, 2021.

More informations: https://firefly.iota.org/

Sending data to the Tangle (2021 version)

In the last version of IOTA, sending data to the tangle is more easier.

```
const { ClientBuilder } = require('@iota/client');

const client = new ClientBuilder().network('testnet').build();

const response = client.message().index('Example').data('Hello from IOTA!').submit();
```

The variable *response* will contain the following structure format:

```
Promise {
 {
   message: {
    networkId: '14379272398717627559',
    parentMessageIds: [Array],
    payload: [Object],
    nonce: '4611686018427618467'
   },
   messageId:
'ad97382c289a53490021188c600c807490432b90ce6728ba8e4ad3395a1491cd'
 }
}
```

Resources

1. https://iota.org
2. Muhammad Salek Ali, Massimo Vecchio, Miguel Pincheira, Koustabh Dolui, Fabio Antonelli, Mubashir Husian Rehnmani - Applications of Blockchains in the IoT: A comprehensive survey, IEEE, 2019
3. https://laurencetennant.com/iota-tools/
4. https://www.iota.org/foundation/research-papers
5. https://docs.iota.org/