Tesi di Laurea Magistrale in Ingegneria Informatica

Car Parking application with IOTA cryptocurrency and Flash Channels technology

Politecnico di Torino

Relatore: Prof. Giovanni Malnati
Candidato: Alessandro Rota

April 2018

Summary

This thesis is aimed at analyzing the possibilities offered by the usage of cryptocurrencies in applications requiring high frequency instant transactions exchange.

My analysis is focused on a particular cryptocurrency, IOTA, based on a new kind of distributed storage called Tangle, that replaces the blockchain used by almost all other cryptocurrencies.

IOTA is born to try solving some problems of the blockchain-based cryptocurrencies, but is heavily inspired by them, in particular by the first and most famous one, Bitcoin.

That's why I constantly compare IOTA to Bitcoin, focusing in particular on two special modules created by these cryptocurrencies to implement the concept of payment channel.

The names of these similar modules are respectively Flash Channels and Lightning Networks.

At the beginning of my thesis I make a detailed introduction to DLT, the special kind of database used to register all transactions of a cryptocurrency.

This introduction is necessary because the main difference between most of cryptocurrencies and IOTA is the novel type of DLT used by IOTA.

After this introduction, I focus on the differences between Bitcoin and IOTA.

Bitcoin uses a blockchain as a DLT. It consists of a sequence of blocks. Each block references the previous one and contains a set of transactions. The block is mined by a so called *miner* by solving a difficult computation that requires much time to be made. The difficulty of this Proof of Work, together with the cryptographic algorithms that are used to link the blocks and the fact that all nodes have their own copy of the blockchain, guarantees that the block, once mined and confirmed by the other nodes, will never be modified or replaced.

Bitcoin in these years has been proven to be working, but has several disadvantages that makes it inconvenient to use.

Users have to pay high fees every time they make a transaction. This makes Bitcoin unusable in case of micro transactions because the fee they have to pay would be higher than the actual exchanged value.

Another issue is the lack of scalability: Bitcoin at the moment can handle just 7 transactions per second (TPS), while other payment systems such as Visa can handle 2,000 TPS or more.

Further critiques are due to the fact that originally every user was able to mine blocks, but nowadays miners are grouped in big mining farms. This could lead to centralization, undermining the reason why Bitcoin was born.

Many of these issues derive from the blockchain technology that is used as a DLT.

To solve them, IOTA founders decided to use a different kind of DLT, called Tangle.

It has been defined as a blockchain "without blocks and without chain". Every transaction approves two previous transactions and is mined directly by its creator with a Proof of Work that is far simpler and faster to calculate than in Bitcoin so that even small Internet of Things devices can issue transactions without any

problem of energy consumption.

This novel structure allows IOTA to scale infinitely. The more transactions are published on the Tangle, the faster they are confirmed.

The second fundamental innovation is that there are no miners, since Proof of Work is negligible and is calculated by the issuer of the transaction. This completely removes fees, enabling use cases involving micro transactions and making IOTA suitable to become the backbone of the Internet of Things and Machine-to-Machine (M2M) economy.

I analyze similarities and differences between Bitcoin and IOTA, explaining in detail how transactions are made and security is achieved.

On top of IOTA, many modules have been built to allow some particular functionality that was not possible to achieve otherwise. These modules extend the basic functionalities of the IOTA protocol and that's why they are collectively known as Iota eXtension Interface (IXI) Modules. I focus on one of these modules, called Flash Channels.

Flash Channels is a bi-directional off-Tangle payment channel that enables instant, high-throughput transactions.

They provide a way for parties to transact at high frequency without waiting for each transaction to confirm on the public IOTA network. Instead, only two transactions will ever occur on the main IOTA network: opening and closing transactions of the Flash channel.

This feature enables new use cases where instant payments are required, such as pay-per-use policies on streaming services.

The participants to the channel are connected end-to-end with whatever technology they like because the communication protocol is not specified by the Flash Channel implementation. Therefore Internet connectivity is not required to use the channel. For example transactions can be exchanged via Bluetooth.

Flash Channels have been released recently and have no documentation explaining how they work on a technical point of view. My work consisted of understanding and documenting the mechanisms underpinning this technology starting from the Javascript code of the open source library provided by the IOTA developers.

To better understand what Flash Channels are, I introduce Lightning Network, that is another similar payment channel developed in Bitcoin. I analyze both technologies and make a comparison between them.

To further demonstrate the potential of this module I realized a simple proof-of-concept Car Parking application using Flash Channels to implement a pay-per-minute policy in a trustless environment.

It consists of a client-server architecture where the client represents the car wallet and the server represents the parking lot.

When the car starts paying for parking, a new Flash Channel is created and all subsequent transactions are not published on the Tangle but are directly exchanged between the parties via TCP connection.

Payments are made once per minute and at every payment the parking meter sends the car a signed ticket valid for a specific minute.

When the car stops the payments, a closing transaction agreed upon by both parties is published on the Tangle and the Flash Channel is closed.

This novel approach to car parking has many advantages.

It allows customers to pay for the effective number of minutes they stay in the parking lot, saving money.

It can be applied to all parking lots, even without gates. This is possible because every car is uniquely identified by its license plate, therefore police officers can easily check if the car with a specific plate is paying using a simple application (not part of this thesis) querying the ticket database.

Customers don't need to come back to the car to manually extend the expiry date of the ticket. They can safely put in the channel an amount superior to what they are actually going to spend because they know the excess money will be immediately refunded when they will leave the lot.

This application can also work in trustless environments, for example when the parking lot is owned by a single entity not trusted by the customer.

Finally I analyze pro and cons of such an application, remarking how it can fit in a wider framework where IOTA becomes the backbone of M2M economy and integrates in all vehicles, forming, together with other Smart Mobility related applications, a whole decentralized and feeless ecosystem.

Ringraziamenti

Desidero innanzitutto ringraziare il prof. Giovanni Malnati per la sua disponibilità, competenza e per l'aiuto fornitomi nella realizzazione di questa tesi.

Ringrazio il centro di ricerca ISMB (Istituto Superiore Mario Boella) per avermi messo a disposizione una postazione di lavoro su cui lavorare al progetto di tesi.

Ringrazio in particolare il Vice Direttore dell'ISMB Edoardo Calia e il dott. Andrea Vesco per i loro consigli, l'aiuto datomi nei momenti di bisogno e per avere offerto a me e ai miei compagni la possibilità di contattare direttamente il team di IOTA, arricchendo notevolemente la nostra esperienza.

Ringrazio i miei compagni di tesi Francesco, Jure, Valentino, e tutte le persone con cui ho condiviso l'ufficio del SiTI che hanno alleggerito le giornate passate a lavorare alla tesi rendendo questo periodo una piacevole esperienza.

Ringrazio infine con affetto la mia famiglia per il sostegno e per essermi sempre stata vicino durante il mio percorso accademico.

Contents

1 Introduction

The main feature that most of cryptocurrencies have in common is decentralization. To achieve this property, it is necessary a data storage with no central administrator.

Therefore, instead of using a traditional database, decentralized storage makes use of a so called Distributed Ledger Technology (DLT).

1.1 What is a Distributed Ledger?

In its simplest form, a distributed ledger is a database held and updated independently by each participant (or node) in a large network. The distribution is decentralized: records are not broadcast to various nodes by a central authority, but are instead exchanged in a peer to peer fashion and held by every node.

Any peer can add records to the ledger. However, records are only accepted when the other peers agree the record meets all the ledger's requirements – typically it must be unique, correctly signed, etc.

The ledger is append-only. It means that data can only be added, not deleted or modified.

In this they are similar to distributed databases (DDBMS): in both cases storage devices are not all attached to a common processing unit such as the CPU, but are spread across several network nodes. Even DDBMS use consensus mechanisms to ensure fault-tolerant communications, and provide concurrency control through locking and/or time-stamping mechanisms [1].

What is the difference between a distributed ledger and a distributed database? The main difference is trust. In DDBMS all nodes are under the control of a single organization. The trust boundary is between the distributed database system as a whole and its users. Instead DLT nodes don't trust each other and so must independently verify data they receive from each other and only share data they are willing to be broadly shared.[2]

DDBMS rely on a central authority to maintain its integrity but with other actors able to request copies and subscribe to updates. All additions and changes to the replicated ledger must be approved by the central authority. While the information in the replicated ledger may be distributed, responsibility for managing it is not.[3]

Distributed ledgers on the other hand, do not rely on an authoritative copy. Indeed, one advantage of distributed ledgers is that anyone who wants to review the ledger can easily obtain a copy, as all copies are equal. The disadvantage is that ensuring the integrity of the ledger is more complicated as we can no longer rely on controlling access to an authoritative, central ledger.[3]

What has been 'distributed' in a distributed ledger is the management of the ledger. This includes deciding what entries to accept, in which order to keep them and ensuring entries once added are not changed. A group of peers shares this responsibility, rather than leaving it to a central authority. With no single agent responsible for maintaining the ledger, we must rely on the consensus of the peers involved. The current state of the ledger is simply the peers' consensus view. Consequently, distributed ledgers aren't defined in terms of how or where the information

is stored, indeed each peer may store ledger data how and where they prefer.[3]

Their main feature is the possibility to work properly in absence of trust among the peers, thanks to the use of a consensus procedure and protocol.

For a DLT to be trusted, two main features must be guaranteed:

1. data haven't been tampered with. This is accomplished using cryptographic functions, such as digital signature and hash

2. all nodes eventually reach the same view of the ledger (eventual consistency)

The best known use of DLT is represented today by the deployment of crypto-tokens or cryptocurrencies.

A cryptocurrency is a digital currency, not existing in physical form, that uses cryptography to secure transactions and a DLT to store them. In this way a trusted third party, such as a bank, is no more needed to guarantee a secure money exchange.

DLT is a general concept that can be implemented in various ways.

The most widely used type of DLT is undoubtedly blockchain. It has been used for the first time in 2008 as backbone of Bitcoin, Ethereum, Ripple and many others. Nevertheless blockchain is not the only possible kind of DLT.

In 2015 one of the first cryptocurrencies not using blockchain has been created. Its name is IOTA and it uses a new kind of DLT called Tangle.

1.2 Blockchain

As said above, blockchain is the most widely used kind of Distributed Ledger Technology. In this section we will refer to Bitcoin blockchain because it is the first one and it is also the starting point for all subsequent implementations.

The blockchain data structure is an ordered, back-linked list of blocks of transactions. The blockchain can be stored as a flat file, or in a simple database. The Bitcoin Core client stores the blockchain metadata using Google's LevelDB database.

Blocks are linked backwards each referring to the previous block in the chain. Those links are usually referred to as "hash pointers".

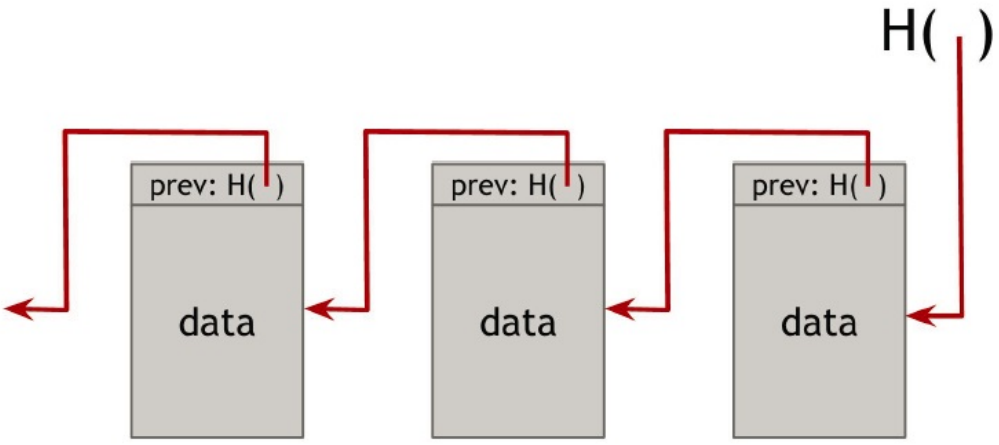

Figure 1: Each block contains a hash pointer to the previous one

The blockchain is often visualized as a vertical stack, with blocks layered on top of each other and the first block serving as the foundation of the stack. The visualization of blocks stacked on top of each other results in the use of terms such as "height" to refer to the distance from the first block, and "top" or "tip" to refer to the most recently added block.

Each block within the blockchain is identified by a hash, generated using the SHA256 cryptographic hash algorithm on the header of the block. Each block references the previous block, known as the parent block, through the "previous block hash" field in the block header. In other words, each block contains the hash of its parent, called "hash pointer", inside its own header. This action of reference is usually called "confirmation". When block B's hash pointer points to block A, it means that block B *confirms* block A.

The sequence of hashes linking each block to its parent creates a chain going back all the way to the first block ever created, known as the *genesis block*.

Although a block has just one parent, it can temporarily have multiple children. Each of the children refers to the same block as its parent and contains the same (parent) hash in the "previous block hash" field. Multiple children arise during a blockchain "fork," a temporary situation that occurs when different blocks are discovered almost simultaneously by different miners. The block discovery process will be explained in the next paragraphs. Eventually, only one child block becomes part of the blockchain and the "fork" is resolved.[4]

1.2.1 Mining and Incentives

In order for a block to be accepted by network participants, the block creator must complete a Proof of Work (PoW) by applying a cryptographic algorithm to all of the data in the block.

The difficulty of this work is adjusted so as to limit the rate at which new blocks can be generated by the network to one every 10 minutes. Due to the very low probability of successful generation, this makes it unpredictable which node in the network will be able to generate the next block and it also limits the rate at which blocks can be created. Computing Proof of Work requires a lot of resources, mainly electric energy and ad-hoc hardware. For this reason a mechanism was introduced to provide incentives for the nodes succeeding in the PoW completion.

There are two main incentives: *block reward* and *transaction fee*.

Block reward consists of a certain amount of bitcoins minted from thin air and assigned to the issuer of the block. The reward is assigned to the miner with an additional transaction included in the block, called *coinbase transaction*.

Due to the analogy with gold mining, people computing proof of work are usually referred to as *miners*.

Block reward is the only way to create new bitcoins. This means that rewards are what makes bitcoin total supply increase. As time goes by, rewards are going to be halved every 4 years until they will become zero.

As a consequence the theoretical supply limit of 21.000.000 bitcoins is supposed to be reached on May 7th, 2140.

After that, no new bitcoins will ever be created and new forms of incentive

will be needed to keep miners working to mine blocks. This is the reason why *transaction fees* have been introduced in the system. At the moment fees are just a complementary incentive but as block rewards will keep decreasing, they will become more and more important.

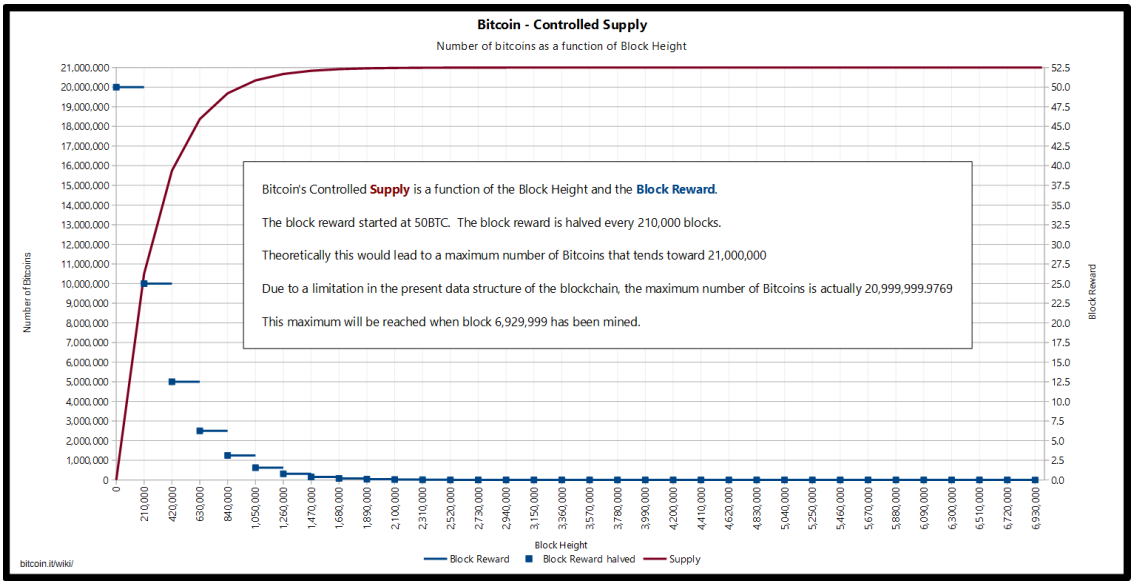

Figure 2: Bitcoin's controlled supply as a function of block height and block reward

Transaction fees is a reward that spenders may include in any Bitcoin transaction. The fee may be collected by the node who wins the mining competition for the block which includes the transaction.

Transaction fees are voluntary on the part of the person creating the bitcoin transaction, as he can include any fee or none at all in the transaction. On the other hand, miners will be incentivized to include a transaction in a block if that transaction has a fee associated.

The transaction fee is therefore an incentive on the part of the bitcoin transactor to make sure that a particular transaction will get included into a block.

When Bitcoin protocol was created in 2008 by Satoshi Nakamoto, Proof of Work was simple enough to be computable with an average CPU. In this way every miner had the possibility to compute its own block, allowing true decentralization to the blockchain.

Since then the Proof of Work complexity has grown up enormously.

Today only mining pools (clusters of servers equipped with ad-hoc ASICs) can successfully participate in the mining competition.

1.2.2 Security

The main property that blockchain should have is data integrity. How to achieve such a property on a database that any node, even malicious ones, can freely read and modify?

When a block is modified in any way, its hash changes. The block's changed hash requires a change in the "previous block hash" pointer of the child block. This

in turn causes the child's hash to change, which requires a change in the pointer of the grandchild, which in turn changes the grandchild, and so on.

This cascade effect ensures that changing the content of a block is very difficult, if not impossible. In fact, once a block has many generations following it, it cannot be changed without forcing a recalculation of all subsequent blocks.

Because such a recalculation would require enormous computation (and therefore energy consumption), the existence of a long chain of blocks makes the blockchain's deep history immutable, which is a key feature of Bitcoin protocol security.

One way to think about the blockchain is like layers in a geological formation, or glacier core sample. The surface layers might change with the seasons, or even be blown away before they have time to settle. But once you go a few inches deep, geological layers become more and more stable. By the time you look a few hundred feet down, you are looking at a snapshot of the past that has remained undisturbed for millions of years.

In the blockchain, the most recent few blocks might be revised if there is a chain recalculation due to a fork. The top six blocks are like a few inches of topsoil. But once you go more deeply into the blockchain, blocks are less and less likely to change.

This is why the transactions included in a block (including the coinbase) can be securely spent only after a certain number of blocks have piled up on top of the corresponding block.

While the protocol always allows a chain to be undone by a longer chain and while the possibility of any block being reversed always exists, the probability of such an event decreases as time passes until it becomes infinitesimal.

Since blockchain is mainly used as a backbone for cryptocurrencies transactions, the most important thing to avoid is a *double spend attack*. A double spend is an attack where the given set of coins is spent in more than one transaction.

For example Mary pays John 10 bitcoins. This transaction gets included in a block, John doesn't wait for enough confirmations and immediately sends Mary the product she paid for. After that Mary could issue a new transaction spending that *same* 10 bitcoins to another address owned by herself.

This transaction of course can't be included in a block confirming directly or indirectly the first transaction, because they are conflicting.

Nevertheless the block containing the second transaction could reference a block *previous* to the block including the first transaction. This forks the blockchain but now both transactions are on the blockchain. The two transactions are conflicting: only one of them will eventually be confirmed.

Nodes usual follow the *longest chain rule*: assuming there is more than one possible path from genesis block to the tips, the new incoming block will probably be appended to the tip belonging to the longest chain.

Obviously miners can decide not to follow the rule and appending the block they mined on a shorter chain, but in this way they would risk to see their block to be never confirmed by the others. If that happens they will have spent computational power uselessly, because they will never be able to spend the reward coming from the unconfirmed block.

For Mary's attack to succeed, the double spend transaction should eventually

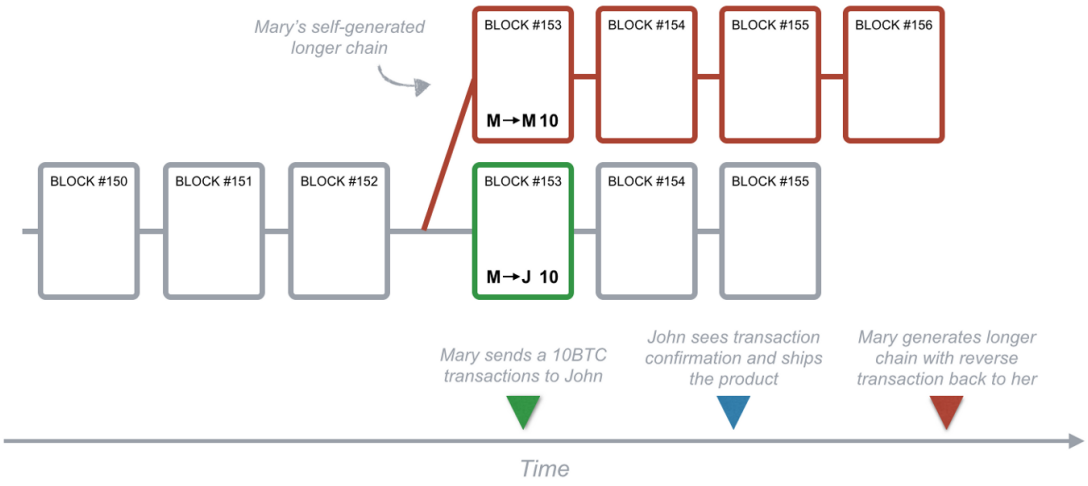

Figure 3: Example of double spending attack

be in the longest chain. In the example above, the original transaction block has been confirmed by 2 blocks. For the attack to be successful, Mary should append to the double spend transaction block at least 3 valid blocks before any other miner appends just one block to the gray block in the figure. If she has enough hash power or luck to do this, the double spend chain (the red one in the figure) becomes the longest one and the attack succeeds.

1.2.3 Problems

Increase of the transaction fees amount

As previously said, block reward decreases in time, and will eventually become insignificant. The only form of reward for the miners will be represented by the transactions fees.

This, together with the constantly increasing value of the bitcoin, is causing transaction fees to rise out of control.

At the days of writing (January 2018) the average transaction fee is more than 20 dollars.

Even if transaction fees are freely chosen by the creator of the transaction, they are usually calculated proportionally to the transaction size in bytes and not to the amount of money exchanged in the transaction, as one could expect.

It means that for a small (in terms of exchanged amount) transaction, the sender will have to pay a disproportionate fee. This makes Bitcoin very disadvantageous to use for small and medium transactions.

Mining pools centralization

Nowadays Proof of Work has become so difficult that no single miner can hope to have considerable profit by mining on his own. That's why they usually gather together in mining pools.

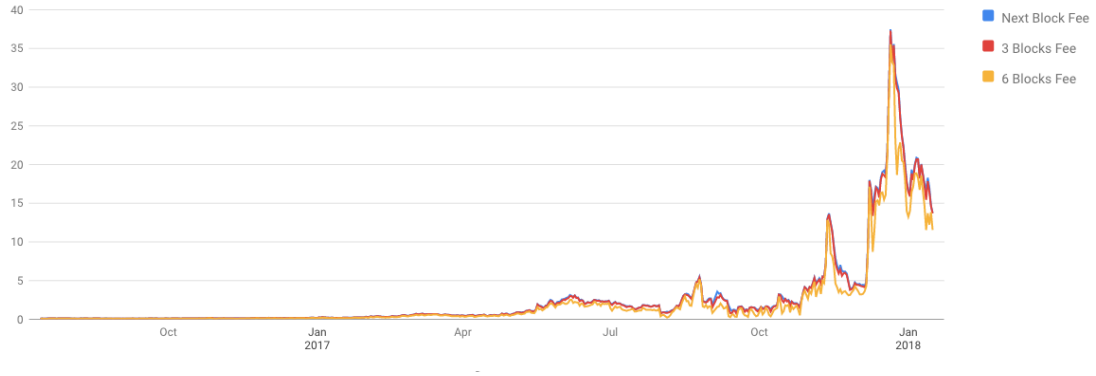

Figure 4: Average transaction fees in US dollars

This creates a threat to Bitcoin: the protocol is born to enable true decentralization in transactions, but mining pools can control large part of the network, introducing a form of centralization.

In June 2014, Ghash.io, the largest mining pool, got so big that it actually had over 50% of the entire capacity over the Bitcoin network.[5]

As a consequence. bitcoin community started fearing the possibility of a *51% attack*.

51% attack is an attack on a blockchain by a group of miners owning the majority of hash power in the whole network. This could lead to reversing transactions that the group sends while it's in control, preventing some or all transactions from gaining any confirmations and preventing some or all other miners from getting any valid block.[6]

Scalability

Scalability is the ability of a system to handle a linearly increasing load with linearly increasing resources.

The most important parameter to measure scalability in a DLT is the number of transactions that it can handle per second (TPS).

How many transactions per second (TPS) can Bitcoin handle?

In the original Bitcoin implementation some parameters are hardcoded. For example block size is 1 MB, the single transaction size depends on the number of inputs and outputs but it has a minimum size and Proof of Work is adjusted so that a block is found every 10 minutes on average. Given those parameters we can calculate a maximum threshold for transaction throughput.

$b = blocksize = 1MB = 10^6\ bytes$
$tx_{min} = minimum\ transaction\ size = 250\ bytes$
$time_{average} = average\ time\ between\ blocks = 600\ seconds$
$tps_{max} = maximum\ number\ of\ transactions\ per\ second$

$$tps_{max} = \frac{b}{tx_{min} \times time_{average}} = \frac{10^6}{250 \times 600} \simeq 7 \tag{1}$$

So how does 7 TPS compare?

It's quite low compared to the throughput of any major credit card processor. Visa's network is said to handle about 2,000 TPS around the world on average, and capable of handling 10,000 TPS during busy periods. Even Paypal, which is newer and smaller than Visa, can handle 100 TPS at peak times. That's an order of magnitude more than Bitcoin.[5]

No Quantum Resistance

Quantum computers are computers which exploit quantum mechanics to do certain computations far more quickly than traditional computers. Those computations could include attacks to Bitcoin private keys associated to pubic keys.

ECDSA-256 - the asymmetric algorithm used by Bitcoin for signature - is not quantum secure, therefore it could be attacked by such computers.

The process of finding a nonce in order to generate a Bitcoin block could be handled very efficiently by a quantum computer, because it is a problem that relies on trial and error to find a solution.

As of today, one must check an average of 2^{68} nonces to find a suitable hash that allows a new block to be generated. A quantum computer would need $\Theta(\sqrt{N})$ operations to solve a problem that is analogous to the Bitcoin puzzle stated above. This same problem would need $\Theta(N)$ operations on a classical computer. Therefore, a quantum computer would be around $\sqrt{2^{68}} = 2^{34} \simeq 17$ billion times more efficient at mining the Bitcoin blockchain than a classical computer.[9]

Humanity currently does not have the technology necessary to create a quantum computer large enough to attack Bitcoin keys or PoW. It is not known how quickly this technology will advance; however, cryptography standards such as ECRYPT II tend to say that Bitcoin's 256-bit ECDSA keys are secure until at least 2030-2040.[7]

Many attempts have been done to try to mitigate or partially solve these issues. This has been done mainly by changing some parameters or part of the original Bitcoin protocol via hard forks, such as BitcoinCash.

Another approach was creating new cryptocurrencies (collectively referred to as *altcoins*) with brand new protocols, always using the blockchain as a backbone, such as Ethereum, Litecoin, Ripple, Monero and many others.

In 2016 the first version of the IOTA project has been released. This cryptocurrency tries to solve some of the Bitcoin issues by changing not only the rules but also the DLT used as backbone. In fact IOTA is one of the first cryptocurrencies that doesn't rely on a blockchain but on a different kind of DLT, called Tangle.

1.3 Tangle

The tangle is a novel kind of distributed ledger architecture that is based on a Directed Acyclic Graph (DAG). One might refer to it as a "Blockchain without Blocks and the Chain" (semantically, it's not really a Blockchain).

Figure 5: The Tangle is a Direct Acyclic Graph

At its core, the Tangle still has the same underlying principles as a Blockchain: it's still a distributed database, it's still a P2P Network and it still relies on a consensus and validation mechanism.

If we look at the blockchain as a graph, every vertex is a block containing many transactions and edges go from the current vertex to the previous one.

Instead in the Tangle every vertex is a single transaction and every transaction references *two* past transactions.

If there is not a directed edge between transaction A and transaction B, but there is a directed path of length at least two from A to B, we say that A *indirectly* approves B.

This referencing of transactions is seen as an approval: with your transaction you attest that those two transactions, and all the transactions referenced directly or indirectly by them are valid and conform to the protocols rules.[8]

The main idea of the Tangle is the following: to issue a transaction, users must work to approve other transactions. Therefore, users who issue a transaction are contributing to the network's security. It is assumed that the nodes check if the approved transactions are not conflicting. If a node finds that a transaction is in conflict with the Tangle history, the node will not approve the conflicting transaction in either a direct or indirect manner.

As a transaction receives additional approvals, it is accepted by the system with a higher level of confidence. In other words, it will be difficult to make the system accept a double-spending transaction.

It is important to observe that the protocol does not impose any rules for choosing which transactions a node will approve. Instead, the creators of the Tangle argue that if a large number of nodes follow some "reference" rule, then for any

fixed node it is better to stick to a rule of the same kind.[9]

The introduction of this new kind of DLT offers advantages that were not possible to achieve with blockchain-based protocols and IOTA, the cryptocurrency using the Tangle as a backbone, exploits those advantages to solve some of the problems of the other cryptocurrencies.

We will see in detail how IOTA works in the next chapter.

2 IOTA

2.1 History

The word "IOTA" refers both to the cryptocurrency and the underlying protocol, including consensus mechanism and transaction making. Sometimes it is improperly used to indicate the distributed ledger, but this is actually the Tangle, as the blockchain is for Bitcoin.

All of the founders of IOTA (David Sønstebø, Sergey Ivancheglo, Serguei Popov, Dominik Schiener), have been in the Blockchain space since 2010 to 2011. IOTA itself came out of a stealth hardware startup, which is working on a new trinary microprocessor with working title 'Jinn'. One of the major differences between IOTA and other projects is that IOTA came out of real necessity.

This necessity deals with one of the most promising trends of technology: Internet of Things (IoT).

Experts estimate that the IoT will consist of about 30 billion objects by 2020. It is also estimated that the global market value of IoT will reach 7.1 trillion dollars by 2020.[10]

According to this data, IoT has a tremendous potential since it's going to be everywhere but it also has a whole range of problems where distributed ledgers could be the solution.

Some example of use cases where IoT and payments could play a crucial role in the future are Machine-To-Machine (M2M) Payments, Security of Things (including identity) and automated execution of processes.[8]

If this scenario realizes, the importance of micropayments will increase in the rapidly developing IoT industry.

One of the main disadvantages of Bitcoin is the high fee that the transaction sender has to pay to see his transaction confirmed in a reasonable amount of time. In the case of microtransactions the fee would be much larger than the amount of value being transferred. This makes Bitcoin completely unusable for this kind of purpose. Furthermore, it is not easy to get rid of fees in the blockchain infrastructure since they serve as an incentive for the creators of blocks.

This leads to another issue with existing cryptocurrency technology, namely the heterogeneous nature of the system. There are two distinct types of participants in the system: those who issue transactions and those who approve transactions. The design of this system creates unavoidable discrimination of some participants, which in turn creates conflicts that make all elements spend resources on conflict resolution.

The aforementioned issues justify a search for solutions essentially different from blockchain technology.

With that in mind, the Tangle was born.

2.2 Advantages over Bitcoin

In the previous chapter we focused on the main drawbacks of using Bitcoin, trying to understand why many people think it cannot be used for the IoT.

Now we will see how IOTA tries to fix this disadvantages.

No Fees

In IOTA there are no miners. Every time a node wants to send a transaction to the Tangle, it must compute the PoW on its own. This PoW is far less difficult than the one computed by miners in Bitcoin. It is more similar to Hashcash, a PoW system used in email spam and denial-of-service attacks. In IOTA it has the further function to prevent Sybil Attacks.

PoW difficulty can be freely adjusted by the issuer of the transaction, provided that it is higher than a certain minimum threshold. Transactions with higher PoW have a lower average confirmation time. That's because all transactions that a node wants to broadcast to its neighbors are put in a priority queue, where priority is given to transactions with higher PoW.

In any case PoW can be accomplished on an average PC in few minutes.

The only cost that a node has to pay to issue the transaction is that of electricity, therefore it is negligible.

IOTA is the first transactional settlement protocol that enables to transact any values in a peer to peer fashion without any transaction fees for either the sender or the recipient. This enables IOTA to be the backbone of all current and future micropayment and nanopayment use cases.

True Decentralization

Lack of miners also means lack of mining pools. There is no more distinction between transaction issuers and approvers. Every node is both able to send transaction and validate transactions issued by others and every node has the same power inside the network.

Infinite Scalability

There are no blocks and there is not waiting time between the issuing of one transaction and that of another one.

Since consensus is parallelized, and not done in sequential intervals of batches as in blockchain, the network is able to grow and scale dynamically with the number of transactions.

Every transaction confirms two other transactions, therefore the more transactions and nodes are in the network, the quicker transactions are verified.

This perfectly fits with the need of IoT sensors to push data and transactions at a very fast rate.

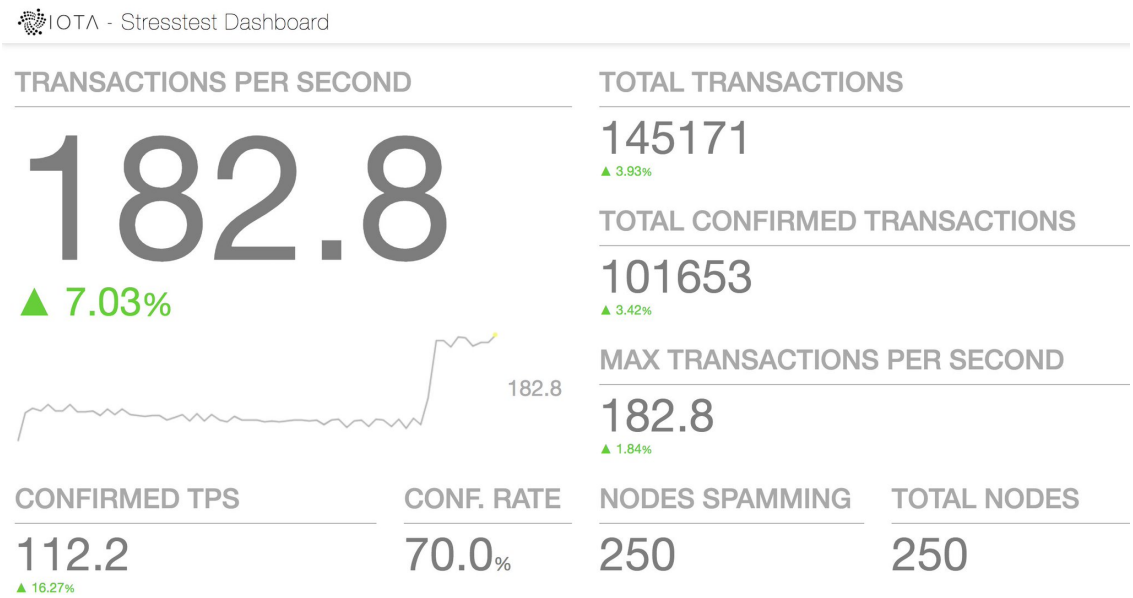

Figure 6: In April 2017 a stresstest made in smaller networks of less than 250 nodes has shown a very promising transaction rate above 100, far higher than Bitcoin.

Quantum Resistance

In IOTA PoW is far simpler than in Bitcoin because the number of nonces that one needs to check in order to find a suitable hash for issuing a transaction is not unreasonably large. On average, it is around 3^8. The gain of efficiency for an "ideal" quantum computer would therefore be of order $3^4 = 81$, which is already quite acceptable.[9]

More importantly, the algorithm that IOTA uses for signature is Winternitz, that is quantum secure.

The drawback of this algorithm is that every time it is used to sign a transaction, it reveals a large portion of the private key. Therefore it is absolutely necessary to change private key every time a new transaction has to be signed.

Offline Transactions

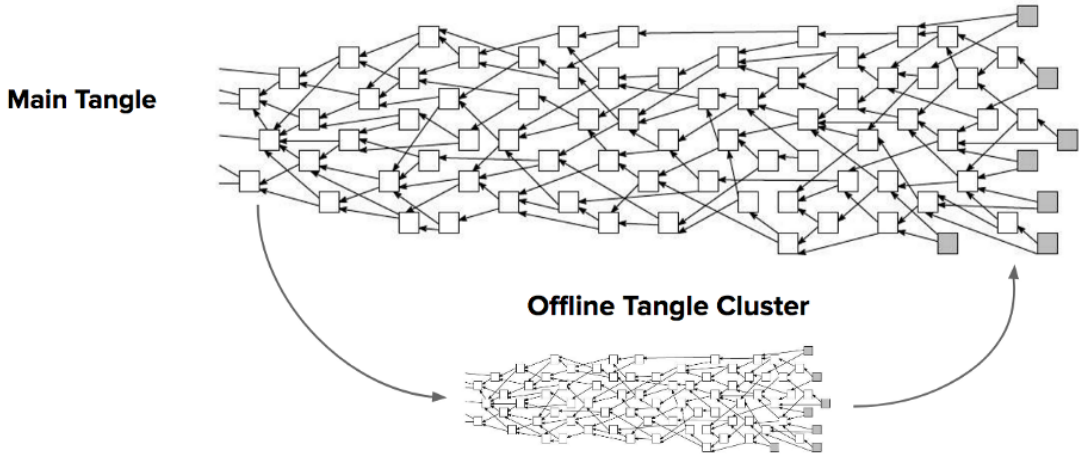

Figure 7: Offline subtangle

In computer science, the CAP theorem states that it is impossible for a distributed database to simultaneously provide more than two out of the following three guarantees:

- Consistency: every read receives the most recent write or an error
- Availability: every request receives a (non-error) response – without guarantee that it contains the most recent write
- Partition Tolerance: the system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes

The Tangle provides A and P. Consistency is eventual because in a specific moment a node is not guaranteed to have the most recent version of the DLT. Some other node could have a new valid transaction that hasn't been broadcast to the queried node yet. Nevertheless eventually all data will be broadcast to all nodes.

The real advantage of eventual consistency is the presence of Partition Tolerance. This allows nodes to work even in offline mode. Any node can create a so called subtangle containing transactions referencing each other but detached from the main Tangle. When the nodes come back online, the subtangle can be merged to the main Tangle if one or more of the subtangle transactions references one or more of the transactions in the main Tangle.

This partitioning is key in being adaptive to the rigorous requirements of an asynchronous IoT environment. There is no such thing as always-on connectivity. An IoT device is not always connected to the Internet. In many cases it could be offline for most of the day and reconnect just in the evening or in a limited interval of time. IOTA makes it possible for a cluster of devices to branch off and still make transactions in an offline environment; utilizing different communication protocols (ZigBee, Bluetooth LE, etc.) for the P2P communication.

This functionality is not present in the blockchain because if a node mines a chain of blocks offline, it will not be able to eventually merge it with the main blockchain. What will happen is a fork: either the new chain or the existing one will be confirmed, not both of them.

2.3 General

Since Bitcoin is the starting point for all the altcoins, many aspects, rules and concepts have been maintained unvaried in IOTA. Therefore we will focus on the differences from Bitcoin and blockchain-based cryptocurrencies in general.

Unlike Bitcoin, IOTA is not inflationary. All IOTA tokens which will ever exist have been created with the *genesis transaction*, which is the first transaction ever created. This means that the total supply of tokens will always remain the same and tokens cannot be mined.

The total supply of IOTA is $(3^{33} - 1)/2$, which equals to a total number of IOTA's of 2,779,530,283,277,761. IOTA is specifically designed for machines, so this high supply makes IOTA optimal for tiny nanotransactions while still keeping efficiency in mind. It also nicely fits into the value in Javascript.

IOTA is based on trinary instead of binary. No official explanation has been communicated for this choice. Anyway, according to the supporters of this decision, motivation seems to rely on the presumed superior performance and minor energy consumption of ternary hardware with respect to binary hardware.[11]

One of the Co-Founder is also involved in a hardware startup called Jinn Labs, which is working on a trinary processor.

The idea is that such processor will be installed in all or most of the physical devices participating to the IOTA network, allowing to compute Proof of Work more quickly.

This choice has consequences on the unit of digital information used to represent keys and addresses. They call this unit *tryte*.

The way trytes are represented is in uppercase latin letters and the number 9 ([9A-Z]). Every seed, key and address is a string that only contains 9A-Z (e.g. 'ABFDSGFDS9').

The *seed* is the master private key. In the wallet released by IOTA, it is the only form of authentication. If someone has access to the seed it can access the account and spend all the money contained in it. Hence it has to be securely stored.

Seeds in IOTA consist of 81 Trytes ('A-Z,9'), which is equivalent to 384 bits security. To make a comparison a Bitcoin private key, which corresponds to a IOTA seed, has 256 bits of security.

Seed is used to derive all the private and public keys by concatenating it with a key index, starting from 0 and incrementing each time a new key pair is created. This allows to create a potentially infinite number of key pairs to be used to sign transactions and validate signatures.

While in Bitcoin the address is the hash of the public key, in IOTA the address *is* the public key.

2.4 Transactions

A transaction is a transfer of IOTA tokens that is broadcast to the network and collected into the Tangle.

Transactions are not encrypted, so it is possible to browse and view every transaction ever collected into a block. Once transactions are buried under enough confirmations they can be considered irreversible.

One first fundamental difference with respect to Bitcoin is the scheme used to guarantee that who spends token actually owns them. Bitcoin uses an Unspent Transaction Output (UTXO) scheme: every transaction is composed by inputs and outputs. Inputs explicitly reference one or more previous unspent transaction outputs. Those inputs represent "what transactions the money come from". All the input values must be spent towards one or more output addresses, which represent "who the money go to".

IOTA doesn't use the UTXO scheme, but a UTXO-like scheme: inputs are not transactions but addresses. This means that we know "what addresses the money come from", not the exact transaction. The task to check whether the specified addresses actually have the spent tokens is left to the node broadcasting the transaction. If the node follows the protocol, it will not broadcast the invalid transaction.

In IOTA the "token transfer object", that is what we usually think of as a transaction, is actually a bundle.

A transfer in IOTA is a bundle consisting of outputs and inputs. Bundles are atomic transfers, meaning that either all transactions inside the bundle will be accepted by the network, or none. A typical transfer in IOTA is a bundle consisting of 4 transactions:

Index	Purpose	Value
0	Output. Recipient of the transaction	> 0 (as defined by user)
1	First bundle entry that spends the entirety of the address input. This bundle entry also contains the first part of the signature (in the example case, it'll be the first half of Alice's signature)	< 0 (spending of input)
2	Second half of Alice's signature.	0
3	Output. If there is a remainder (Alice didn't spend her entire balance at the respective key index), it will be sent to a remainder address.	> 0 (input - output)

A single transaction can contain multiple input and output transactions.
A transaction in IOTA consists of 2673 trytes (if encoded).

Field	Description	Type	Length
*address	In case this is an output, then this is the address of the recipient. In case it is an input, then it is the address of the input which is used to send the tokens from (i.e. address generated from the private key)	String	27-trytes
attachmentTimestamp	Timestamp after POW	Int	
attachmentTimestampLowerBound	Lower Bound of timestamp	Int	
attachmentTimestampUpperBound	Upper bound of timestamp	Int	
branchTransaction	Transaction being approved	String	81-trytes
bundle	bundle hash, which is used for grouping transactions of the bundle together. With the bundle hash you can identify transactions which were in the same bundle.	String	81-trytes
*currentIndex	Index in Bundle	Int	
hash	Transaction hash	String	81-trytes
*lastIndex	Last index in Bundle	Int	
nonce	The nonce is required for the transaction to be accepted by the network. It is generated by doing Proof of Work (either in IRI via the attachToTangle API call, or with one of the libraries such as ccurl).	String	27-trytes
*obsoleteTag	User-defined tag (removed soon)	String	27-trytes
signatureMessageFragment	Signature message fragment. In case there is a spent input, the signature of the private key is stored here. If no signature is required, it is empty (all 9's) and can be used for storing the message value when making a transfer.	String	2187-trytes
tag	User-defined tag	String	27-trytes
*timestamp	Timestamp (not-enforced, can be arbitrary)	Int	
trunkTransaction	Transaction being approved	String	81-trytes
*value	Transaction value	Int	

[12]

Fields marked with "*" compose the so called *essence*. The essence part of all transactions is used to calculate the bundle hash and put it in the *bundle* field of every transaction of the bundle.

This is the current anatomy of a transaction, but it's going to be changed soon.[14]

Some fields, such as *attachmentTimestampLowerBound* and *attachmentTimestampUpperBound* have been recently added to try to enforce a confidence intervals for timestamps.

Timestamps are very important in DLT because they enable interesting features such as smart contracts.

Unfortunately the Tangle is a graph with only a partial order structure, which makes it difficult (in fact, generally impossible) to establish the correct time order of transactions.

Even if all transactions have timestamps on them, we cannot be sure that all these timestamps are accurate (there can be some malicious nodes that want to fool the network about the true time when their transactions appear, and/or some nodes with a wrong clock).

Nevertheless, one can determine the confidence intervals for timestamps with reasonable accuracy.[15]

According to the developers, *obsoleteTag* will be removed and the final bundle essence fragment will be composed by:

Field	Length
*address	27-trytes
*attachmentTimestamp	9-trytes
*attachmentTimestampLowerBound	9-trytes
*attachmentTimestampUpperBound	9-trytes
*bundleNonce	27-trytes
*extraDataDigest	81-trytes
*value	9-trytes

In particular the introduction of *extraDigest* field will make it possible to store immutable data in a transaction. At the moment this is not possible because *signatureMessageFragment* is not part of the essence. Therefore, as long as the transaction is not confirmed, anybody could replay/reattach it with a different content in the *signatureMessageFragment* field and the luckiest transaction would win. Instead *extraDigest* is part of the essence, so setting the field value to the hash of *signatureMessageFragment* would allow to detect cases of changed content.[16]

To better understand how to make a transaction in IOTA, let's consider this scenario: Alice wants to send 15i to Bob.

Alice's addresses:

- A1 balance: 10
- A2 balance: 10
- A3 balance: 0

Bob's addresses:

- B1 balance: 0

These are the main steps to create the bundle:

1. **Make transaction bundle**

   An output transaction is created with a positive value of 15i and address: Bob's address.

   As we can see, none of Alice's addresses contains enough tokens to make the transaction, so it is necessary to have 2 input transactions. Each input transaction has a negative value because it subtracts tokens from an address and must contain the sender's signature to guarantee that who is subtracting tokens is actually the owner of the address.

   Every address is composed by a variable number of trytes, according to its security level. Default level is 2, then for each input transaction, a further meta-transaction is needed to contain the second part of the signature. This is a normal transaction with 0 value.

   Every input transaction subtracts all tokens from the specified address. As a consequence, in our scenario there will be 20i as input and 15i as output.

   Every bundle must be balanced: it means that the sum of input transaction values must be equal to the sum of output transaction values. Then one further output transaction must be added sending the remaining 5i to another of Alice's addresses.

2. **Finalize bundle**

   The essence part of all transactions is used to calculate the Bundle Hash using Kerl hashing algorithm. Then the bundle hash is inserted into all transactions of the bundle.

3. **Add signature for input transactions**

   In IOTA the private/public key pair is generated deterministically from the seed + key index.

   Transaction signature is obtained from a private key and the bundle hash. The signature is added to the Signature Message Fragment field of the transaction.

4. **Get 2 tips**

   This is the first thing requiring to be connected to a IOTA full node.

   Every transaction must reference 2 tips to be considered valid. Referencing means containing the transaction hash of that transaction. The referenced transactions are called trunk and branch.

   The selected tips must not be in conflict with each other and the algorithm to choose them is the so called MCMC.

The interesting thing is that a bundle is composed by N transactions, each referencing 2 tips. So in total each bundle references 2*N transactions, not just 2.

Nevertheless the current implementation of the tip selection algorithm returns only 2 transactions to reference.

The other (2*N - 2) tip are selected from *inside* the same bundle, as shown in the figure below.

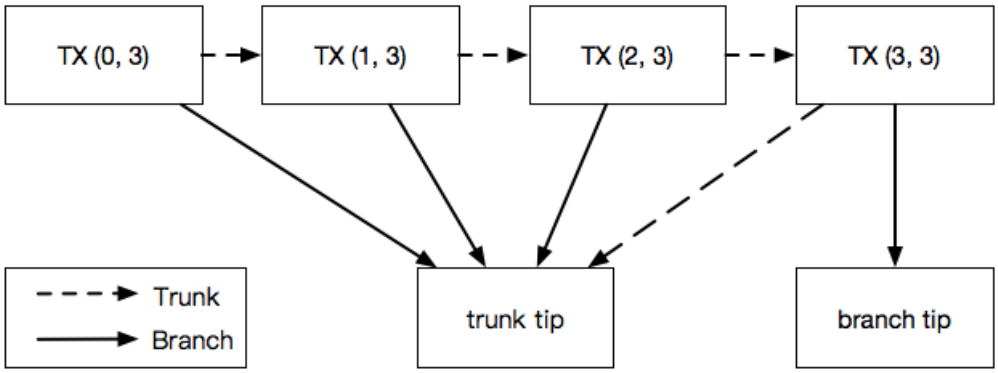

Figure 8: TX(currentIndex, lastIndex). TX are the transactions belonging to the same bundle. As we can see the "external" referenced tips (trunk and branch tip) are just 2. One is referenced 4 times, the other one just 1 time

In this way by following the chain of trunkTransaction references starting from TX(0, 3) we can obtain the whole correct bundle.

In the future things might change, for example by allowing a bundle composed by N transactions to confirm N+1 external transaction instead of just 2.[13]

There are going to be changes also in the transaction fields.

5. **Proof of Work**

   Proof of Work is accomplished for each transaction of the bundle. The obtained nonce and transaction hash are added to the transaction field.

   If anyone takes the transaction concatenated with the nonce and calculates the hash, it should obtain the transaction hash specified inside the transaction. If the obtained hash is different, it means that the transaction has been modified.

   POW can be done locally on the sender's device or delegated to a IOTA node the device is connected to. It should be noticed that most of the public nodes forbid POW delegation towards them to avoid resource consumption.

6. **Bundle broadcast**

When the transaction is complete, it is sent to a full node. This node will broadcast it to its neighbors, the neighbors will broadcast it to theirs and so hopefully the transaction will quickly reach most of the nodes.

Every honest node doesn't propagate transactions that are considered invalid according to certain criteria. For example transaction with insufficient PoW will be rejected.

Note that this criteria are not the same on which node decide whether to approve or not a transaction by referencing it as branch or trunk of a new transaction!

Many kinds of invalid transaction can be broadcast by honest nodes, but they will never be confirmed because they will never be chosen by the MCMC algorithm.

Some examples of invalid transaction that are propagated but will never be confirmed are:[17]

- A transaction whose bundle hash does not match the (actual recomputed) hash of the complete bundle
- A transaction whose bundle indexes do not match other transactions in the bundle
- A transaction whose bundle's total sum is not zero
- A transaction which spends IOTA but where the signature (possibly spread over more than one transaction linked by it) does not match the address the IOTA is spent from

2.5 Consensus

With the word "consensus" we mean the process through which all nodes agree on the same state of the DLT.

As previously said, one of the main differences between blockchain-based cryptocurrencies and IOTA is the way consensus is achieved.

Instead of a smaller subset of the network being responsible for the overall consensus (e.g. miners for Bitcoin, stakers for Ethereum), the entire network of active participants (i.e. devices making transactions), are directly involved in the approval of transactions. As such, consensus in IOTA is no longer decoupled from the transaction making process: it's an intrinsic part of it, and it's what enables IOTA to scale without any transaction fees.

Before starting to see in detail how consensus works, a little bit of nomenclature.

A transaction *approves* another transaction by referencing it directly or indirectly. Approval means making sure that the referenced transaction is valid according to the IOTA protocol rules.

Indirect approval means that the referencing transaction didn't choose the referenced transaction as branch or trunk, but there is a finite path from the referencing transaction to the referenced one.

A *tip* is a transaction that has not been approved by any transaction yet.

A transaction is said to be *confirmed* when it is referenced directly or indirectly by *all* of the tips.

Another important concept is *weight*: every transaction has its own weight that is proportional to the amount of Proof of Work that the issuing node invested into it.[9]

The current implementation sets the own weight of any transaction to 1 for reasons of simplicity. Future plans might introduce a more sophisticated weight calculation algorithm, but this is still part of on-going research.[18]

The *cumulative weight* of a transaction is defined as the own weight of a particular transaction plus the sum of own weights of all transactions that directly or indirectly approve this transaction.

MCMC Algorithm Markov Chain Monte Carlo is the algorithm used to select the two tips to reference. It plays a fundamental role in the protocol because security, double spending protection and confirmation depend on the choice of the "right" tips.

Without going too much in mathematical detail, the idea is to place some particles, called random walkers, on some transactions of the Tangle (possibly quite deep in the DAG so that those transactions can be considered reasonably confirmed) and let them walk towards the tips in a pseudo-random way. The tips "chosen" by the walks are then the candidates for approval.

At every step the walker has to choose which transaction to jump to. The candidate set is composed by the transactions that directly reference the current transaction.

The choice is pseudo-random because it's not deterministic, but the probability of choosing one particular transaction heavily depends on the cumulative weight of such transactions.

Let's suppose the walker is on transaction x. The set of transactions to choose from as next step for the walker is indicated as z and it is the set of transactions approving x. Taking a particular transaction y from this set, the transition probabilities of the walkers are defined in the following way: if y approves x ($y \rightsquigarrow x$), then the transition probability P_{xy} is

$$P_{xy} = \exp\big(-\alpha(\mathcal{H}_x - \mathcal{H}_y)\big)\Big(\sum_{z:z\rightsquigarrow x} \exp\big(-\alpha(\mathcal{H}_x - \mathcal{H}_z)\big)\Big)^{-1} \tag{2}$$

Besides the complexity of the equation, it simply means that the walker heavily tends to choose the transaction with the minimum cumulative weight gradient between the current transaction and the potential next transaction.

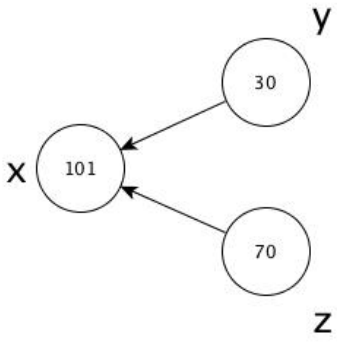

Figure 9: Each circle is a transaction. The number inside each transaction is the cumulative weight of the transaction.

For example in the figure above the walker is on x and has to choose among the set of transaction approving it. This set is composed by z and y. Let's calculate the probability of each choice supposing for simplicity $\alpha = 1$.

$$P_{xy} = \exp\big(-(101-30)\big)\Big(\exp\big(-(101-70)\big) + \exp\big(-(101-30)\big)\Big)^{-1} =$$
$$= \exp(-71)\Big(\exp(-31) + \exp(-71)\Big)^{-1} \approx 4.25 * 10^{-18}$$

$$P_{xz} = \exp\big(-(101-70)\big)\Big(\exp\big(-(101-70)\big) + \exp\big(-(101-30)\big)\Big)^{-1} =$$
$$= \exp(-31)\Big(\exp(-31) + \exp(-71)\Big)^{-1} \approx 1$$

As we can see, the probability of choosing y as next step is negligible. That's because z has already been approved by many transaction, hence it has high cumulative weight, while y has been approved by few transactions, hence it has low cumulative weight.

This is somehow similar to blockchain, where to decide to which branch attach the new block, nodes usually choose the longest branch.

Since nowadays every transaction has own $weight = 1$, it looks clear that the walker tends to walk towards transactions that have more approvals.

Tangle State In contrast to blockchain technologies, IOTA does not build a clocked sequence of static blocks, each one containing a number of transactions. Instead, every single transaction can be attached to the Tangle by itself and in parallel to other transactions.

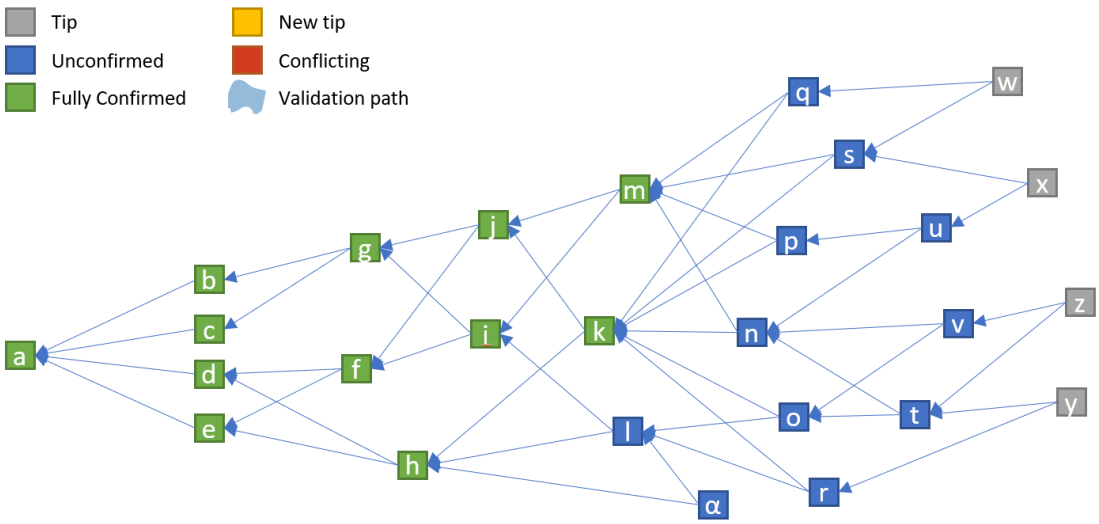

Figure 10: Tangle sample

The graph above shows a sample of a Tangle which will be used to walk through some examples. In this and subsequent examples, green transactions are already confirmed by the network with high certainty, while the blue ones are only partially confirmed (with lower certainty). The gray (and later "yellow") boxes represent tips without any validation. Red transactions, on the later images, are conflicting or invalid ones.

It's important to remark that, as with blockchain, the concept of 100% carved-in-stone certainty doesn't exist. Every transaction has a confirmation probability that can change in time, increasing but also potentially decreasing. When we consider a transaction as confirmed, it means that the probability that it becomes unconfirmed in the future is negligible, but it will never be zero.

In the graph above transaction "α" is an example of an unusual transaction. It is referencing transaction "h" and "l". Since transaction "h" is already referenced by transaction "l", "α" would select one tip "l" and one transaction that is obviously no tip at that time anymore "h". Such behavior seems to be no issue and tolerated by the network, currently.

Adding a transaction In order to add a new transaction to the tangle, a user has to randomly pick two tips (yet unconfirmed transactions) and validate them. Validating means that the user checks the tip's signature, its PoW and makes sure that the tip is not in conflict with any of the previous (directly or indirectly referenced) transactions. If the chosen tips are legit, the user adds its new transaction by referencing them.

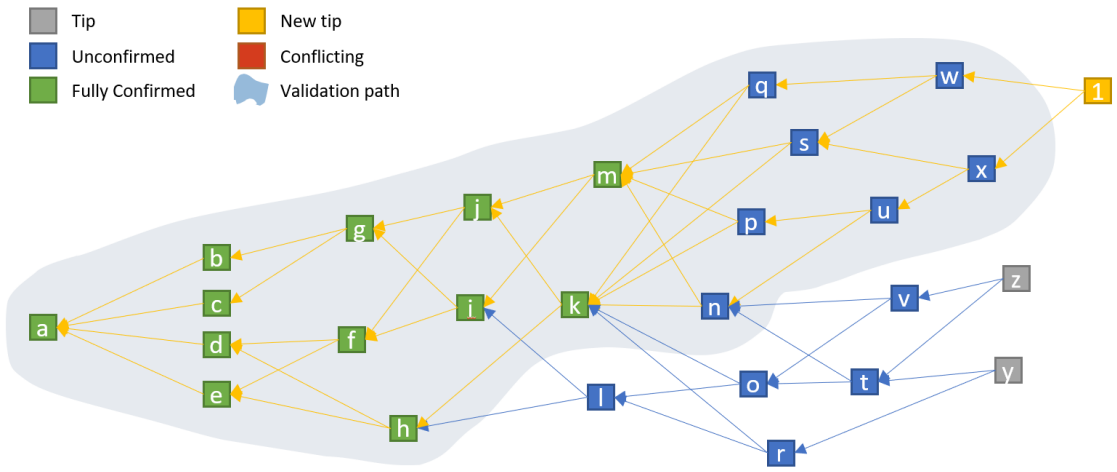

Figure 11: Tangle with new transaction

The node issuing the transaction *only* takes care of validating the subtangle composed by the transactions referenced directly or indirectly by the chosen tips (gray area in figure 11). Inside this subtangle there must not be conflicting transactions, but there could potentially be conflicts between transactions inside the subtangle and transactions outside of it. Nevertheless transactions neither directly nor indirectly referenced by the chosen tips are irrelevant for the current validation process. Somebody else or a later transaction will take care of validating and knitting them into the Tangle. As other users select and validate different tips and paths, a collaborative validation of the complete Tangle emerges.

Adding another transaction At the same time (or earlier or later, whatever) another user might be about to add its new transaction in a different position. It chooses the tips “z” and “y”. By doing so, it is validating a large portion of the same transactions as already validated via transaction "1“ (“a” to “k”, “m” and “n”), plus some additional ones that were not in the validation path of transaction "1“ (“l”, “o”, “r”, “t”, “v”, “y” and “z”).

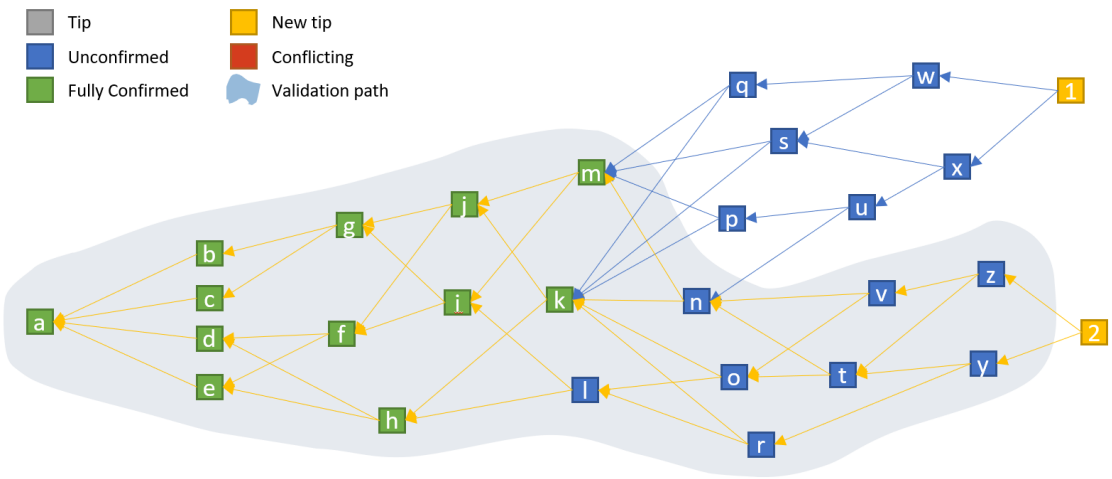

Figure 12: Tangle with another new transaction

New Tangle State By overlaying the validation paths of transaction "1" and "2", we can see that some of the transactions are only approved by one, while others were approved by both of them. Transactions validated and approved by all of the current tips are considered fully confirmed. Hence, transactions "n" moved deeper into the Tangle and turned green now. All subsequent transactions attaching to "1" and/or "2" or its children will keep re-validating and approving transaction "n" from now on.

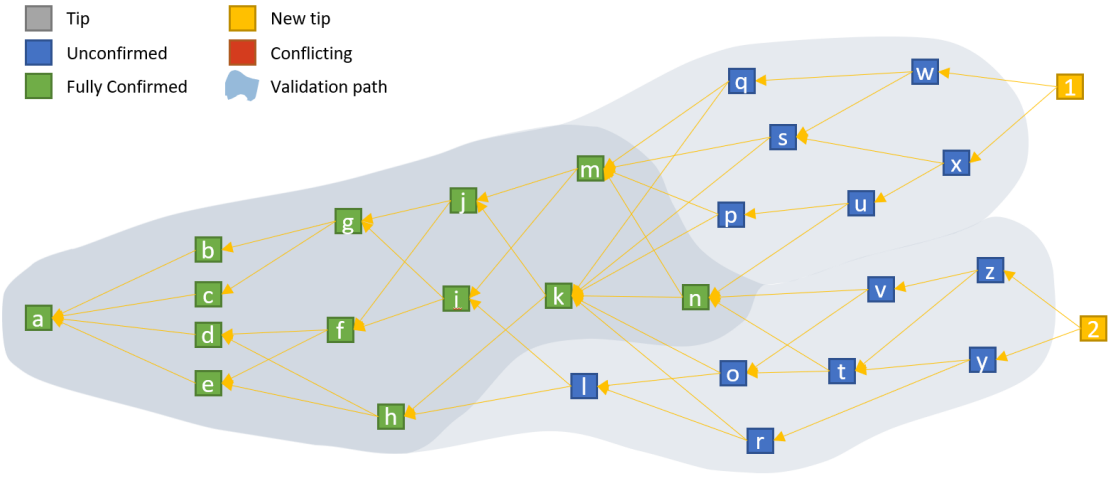

Figure 13: Tangle state after addition of the two transactions

In order to check for confirmation, a recipient only needs to check whether the transaction is directly or indirectly referenced by all available tips (or by a certain rate of them if a lower certainty, such as 80%, is accepted). No re-validation or similar is necessary.

Note: there might be thousands of tips. Instead of checking the parents of each of them, it is possible to select a random sample and do a statistical evaluation.

Confirmation Levels As previously said, complete and immutable certainty of the validity of a transaction can never be achieved. As a consequence, the acceptable confirmation level is decided by the merchant. For example, when selling a low price product needing to be shipped quickly, the merchant could accept a level of confirmation of 75%. For bigger transactions instead it could prefer waiting until 90% confirmation level is reached.

Propagation Delays Nodes are connected via Internet and this inevitably means that there could be propagation delays. Not all nodes see the same Tangle in the same moment.

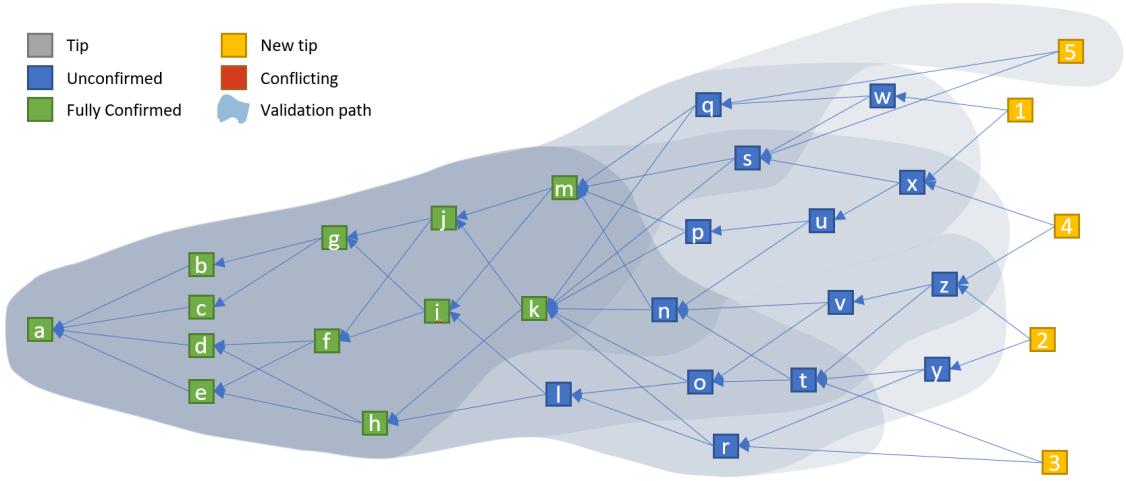

Figure 14: Tangle state after addition of the two transactions

For example in figure 14 "q" and "s" are not tips anymore and new transactions should not directly reference them by using MCMC, even if there is no enforcement on that. But even using MCMC, another node could have a not fully updated version of the Tangle and still see "q" and "s" as tips. Then this node could choose that very tips as branch and trunk. This is what happens with transaction "5".

After transaction "5" is broadcast to the rest of the network, transaction "n" is not fully confirmed by all tips anymore. However, its confirmation certainty is still quite high with 4 out of 5 tips (in reality there would be thousands instead of 5 tips). It's all about a high probability of certainty, just as in blockchain technologies where each block on top increases the probability of certainty.

Please note that transaction "5" in this example is *not* flipping the transaction's state from "confirmed" back to "unconfirmed". It just changes the mathematically exact number of certainty (e.g. if there were 100 tips in total, from 100% to 99%). Once some subsequent transaction references e.g. transaction "1" and "5", transaction "n" will be fully confirmed by all tips again. Such minor confirmation level variations will even less likely happen, the further transactions move into the Tangle.

A confirmation level of 100% might be hard to achieve anyways, as there could always be some lazy tip around (e.g. referencing something useless or not following the protocol).

Double Spending The main problem every cryptocurrency has to solve is double-spending.

Since cryptocurrencies are not physical money, nothing prevents a user from sending a merchant some tokens and then sending *that same* tokens to another merchant.

Of course not both transactions can be considered valid: one will be confirmed and the other one will be considered invalid. That's what we previously called *conflicting transactions*.

For example the confirmed transaction could end up being the one sent to the second merchant.

If in the meanwhile the first merchant has already given to the sender the service or product he made the transaction for, the merchant will never be able to spent the tokens from the transaction he received because it's now invalid.

That's why it's so important to manage double spending and providing the users secure methods to consider a transaction as confirmed.

It's interesting to note that double spending has a slightly different meaning in Bitcoin and IOTA.

Bitcoin uses a UTXO scheme for transactions: it means that every transaction spends tokens received in one very specific transaction in the past. Double spending happens when a transaction "A" spends tokens coming from a transaction "B" whose outputs had already been spent previously.

This doesn't mean that the sender address has not enough tokens for issuing transaction "A". Maybe the sender address has received many other transactions and has enough tokens to fund transaction "A". But the point is that "A" spends tokens that have already been spent.

Conflicting transactions will be conflicting forever and inevitably all but one will eventually result as invalid.

Instead IOTA uses a UTXO-like scheme where transactions don't take tokens from a specific transaction, but rather from a specific address. This means that double spending consists in making a transaction using as input an address on which there are not enough tokens for that transaction.

If in the future that address will receive enough tokens to be able to fund the transaction, it will not be a double-spending anymore.

Hence conflict could simply be a temporary situation that will be resolved after some time.

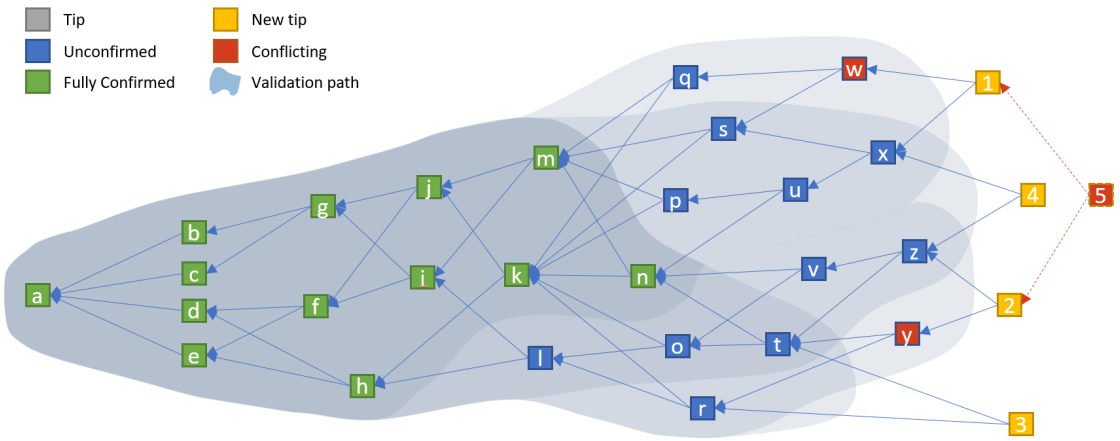

Figure 15

Imagine a situation where a user adds two conflicting transactions in different areas of the Tangle ("w" and "y"). Subsequent users might only have one of these conflicting transactions in their validation path (based on their tip selection and maybe due to propagation delays). For example, the users attaching transactions "1" and "2" will not see the conflict and confirm their chosen tips. Hence, the double spend attempt got its first confirmations. However, sooner or later it must happen that both conflicting transactions are in the path of validation of one transaction. For example, transaction "5" would see the conflict and not attach to the selected tips. Instead it will reselect tips until it will find two not conflicting ones in order to be sure its transaction will be considered valid.

Depending on the tip selection and Tangle progress, it might happen that many more users attach their transactions behind "w" *or* "y" (never both), before the conflict becomes clear. Depending on where users attached most new transactions, either "w" or "y" will confirm at some point, while the other gets abandoned. All subsequent transactions attached to the abandoned one (as they couldn't see the conflict coming) will also be abandoned. However, they are not lost but can be taken by anybody (but most likely the payment recipient) and reattached to the Tangle for a new chance of confirmation. The PoW will need to be redone, but no fresh signatures from the sender will be required.

Double Spending Resolution In the previous figure a user tried to connect a transaction "5" to the tips "1" and "2". Due to a conflict it retried the tip selection, and decided to attach to tips "1" and "4".

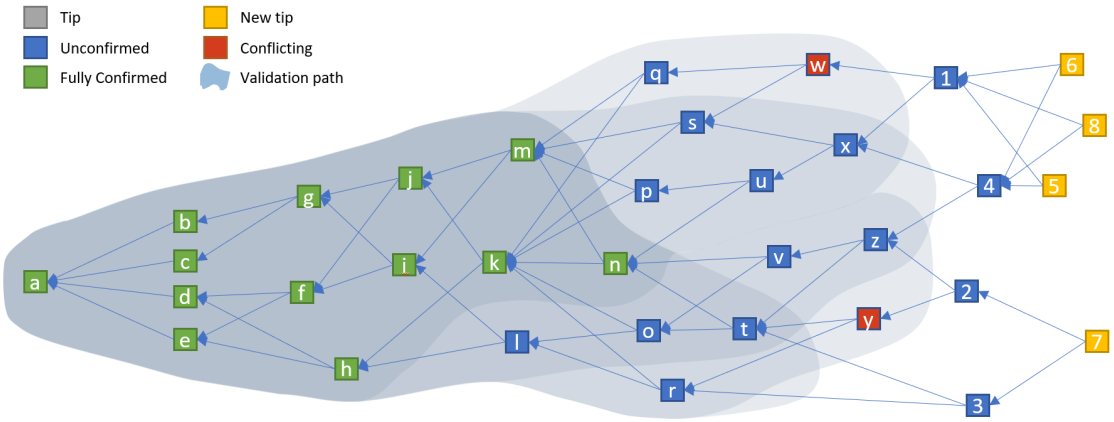

Figure 16

Another user (or maybe the same one) chose tips “2” and “3” to attach transaction “7”. Some kind of branches emerged, but only one can survive, due to the double spend in “w” and “y”. Based on the random selection of tips (and the cumulative weight of the transactions), one of the two branches will receive more child transactions (respectively, weight) until the Tangle will turn into a state where it will not be possible to attach legitimately to one of the segments anymore. In the sample above users could just continue attaching to transactions “5”, “6” and “8” but not to transaction “7” anymore. Hence, transactions “y”, “2”, “3” and “7” will never make it into a fully confirmed state.

As described in the previous figure, transactions “y”, “2”, “3” and “7” could be reattached to the Tangle again. As long as they are (still) valid, they have a fresh chance of confirmation. Transactions “2”, “3” and “7” might become confirmed then, while transaction “y” will stay invalid.

Offline Tangle That’s one of the most peculiar features of the Tangle.

The Tangle allows users to continue building transactions while they are offline (e.g. within a company intranet environment, or to continue interacting with neighbors during an Internet outage). To do so, transactions are created and connected to each other as described by the protocol.

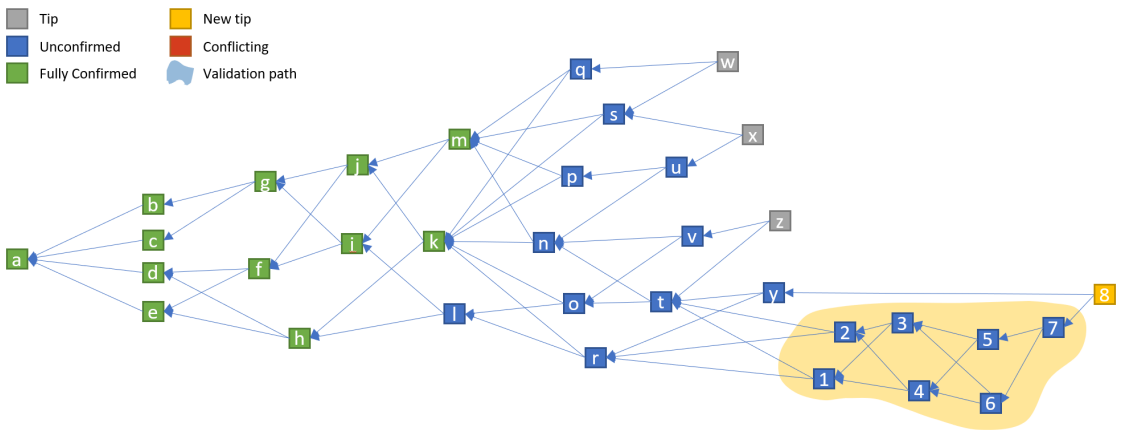

Figure 17

In the figure above, transactions “1” and “2” are the first offline ones. They are connected to the last known tips of the online Tangle. Subsequent transactions attach as usual. Once a commit to the main Tangle is desired (or possible, in case of an Internet outage), the offline subtangle is finalized by creating transaction “8”, which is merging the offline Tangle with a recent tip of the online Tangle.

Transaction "8" is not necessary because the subtangle is already connected to the main one by transaction "1" and "2", but it’s really recommended. The reason is that if the subtangle has been offline for a long time, it is very unlikely that its transactions will be confirmed, because the tip selection algorithm favours recent tips. Subtangle tips (transaction "7" in this case), regardless of when have been made, "look" very old to the algorithm because the connection to the main Tangle is very old.

So transaction "8" is useful to promote the subtangle and increase the chances to be confirmed.

Subsequently, transaction “8” turns into a legit tip and can be selected and validated by later online transactions. The next users attaching to transaction “8” online will include all offline transactions in their validation routine.

Please note that offline transactions can only become fully confirmed once they made it into the main Tangle like any other transaction, just as shown on the previous slides. If any transaction within the offline branch was conflicting with the main Tangle, transactions “1” to “8” would not become confirmed. Again it might take some subsequent transactions until the conflict is visible from all (or the majority) of the main Tangle’s tips.

2.6 Security

In the previous section we started to see how consensus protocol guarantees security and immutability to data stored in the Tangle.

Now we will deep dive a bit more in the description of possible attack scenarios, trying to explain why they are not going to succeed in IOTA.

Security is only intended as DLT security. Many other kinds of attack are possible like key stealing from the users, thefts and hackings on exchange platforms,

but those are not due to weaknesses in the consensus protocol, therefore they will not be the object of the discussion.

Like any other cryptocurrency, all security threats and solutions come from a common problem: guaranteeing the irreversibility of transaction by avoiding double spending.

For example the consensus model in Bitcoin relies on PoW, mining, and block validation by the nodes in the network.

It's mainly based of the difficulty to compute PoW. A single malicious user or group of users is not supposed to be able to successfully computing PoW faster than all the rest of the network put together.

This system has been proved to be working but some kind of attacks are possible anyway.

The most well-know is 51% attack: if attackers got control over 51% of the hash-rate they would be able to double spend as they like, because since the attacker can generate blocks faster than the rest of the network, he can simply persevere with his private fork until it becomes longer than the branch built by the honest network.

No amount of confirmations can prevent this attack; however, waiting for confirmations does increase the aggregate resource cost of performing the attack, which could potentially make it unprofitable or delay it long enough for the circumstances to change or slower-acting synchronization methods to kick in.

Research has been done in the years since that theoretical attack was postulated and it was found that the percentage necessary to conduct this kind of attack is actually lower: it would be sufficient around 34% of network hashing power to carry out the attack.

Built on a directed acyclic graph, the Tangle, IOTA has a few more differences to blockchains and their field of application.

The most notable ones are the fact that there are no miners, no blocks, no constant hash rate and PoW is far less complex than the Bitcoin counterpart.

Consensus lies solely at the users, that have to approve two other transactions before they can send one.

Let's move to how 34% attack applies to IOTA.

The most crucial step to understanding all of this is that IOTA network has a mesh net topology. This differs greatly from all other blockchain protocols. Mutual tethering and the future of IoT connectivity are the factors that make IOTA a mesh net, which has some implications for network security, the most important of which is how this topology strengthens network resiliency against the 34% attack.

IOTA being a mesh net means that each full node only sees a tiny part of the Tangle through their handful of neighbors. No one has a list of all IPs of all nodes.

Because blockchains are not in mesh nets, the 34% attack in blockchains just means that if an attacker gets enough hash power, he can successfully conduct the attack. Percentage of network hash rate is the only variable in the block chain 34% attack. However, in IOTA, there are three variables required for this attack.

1. **X percentage of network hash rate** A sufficiently large portion of the network hashing rate. Just like in Bitcoin, the attacker would have to achieve

a certain very large amount of network hashing power in order to overtake the network. But this is not the only requirement in IOTA.
Moreover there is not an "all or nothing" network takeover in IOTA. Without going too much into detail, let's say that 34% attacks only take down layers of the Tangle, requiring an exponentially stronger three variable attack to propagate deeper into the Tangle.

2. **Omnipresence** This is not mandatory, but to deploy attack resources properly and efficiently, an attacker would need to get a broad overview of every full node connection in the Tangle to optimize the attack propagation. This is impossible since every connection is kept private, and no entity is able to map the network. At the moment the only way for a full node to add neighbors is by asking to other community members on a chat application. It means that no nodes can have a full list of the available nodes.

3. **Y percentage of omnipotence** Neighboring with a sufficiently large portion of the network (Y% omnipotence). This is necessary because of the bandwidth limitations that are assumed to be present in a mesh net composed mainly by IoT devices. The attacker must be able to push their massive amount of hashing power (X% of the network's hash power) through the Tangle suddenly, but with a limited number of neighbors, it will be able to send its massive amount of hashing power to just a small part of the network.
In other words an attacker can create a massive amount of transactions confirming his double spend, but he will never be able to spread them across the network quickly enough because bandwidth and limited number of neighbors act as bottlenecks.

Let's say that X = 25% and Y = 15%: The attacker would bring down a small number of "edge nodes" (the nodes that the attacker is connected to).

The combination of X and Y will determine what percentage of edge nodes are taken down in the attack, and thus the effectiveness of the attack. X can be 99%, but without sufficient Y, the attack can only bring down a very small percentage of edge nodes (around Y%). The edge nodes and nodes surrounding some of those edge nodes would be overwhelmed with the attack and restart or just blacklist the attacker so that their nodes can become functional again. The low latency nature of a mesh net means that there is a gradient of attack that depends on X and Y.

These are the main reasons why the tiny amount of PoW required by IOTA to issue transactions is not a threat to its security.

2.7 IXI Modules

The core IOTA protocol is purposefully lightweight and simple in nature. Its role is to enable trustless and feeless transactions as well as tamper proof data transmission.

The engineering philosophy behind IOTA is one of modularity and pragmatism. Instead of creating "all in one" solutions that could lead to mediocre performance in each feature, IOTA decided instead make its platform modular.

This means that the underlying ledger protocol should remain as basic and efficient as possible and extend its utility through the so called Iota eXtension Interface (IXI).

An IXI Module is a 2-layer stand alone component that is optimized for its purpose, without trading off performance or functionality for other applications.

This gives users of IOTA the choice to tailor their usage of the protocol, instead of imposing the overhead of the features on, even if they originally only need to use one of them for instance. Both of these components are imperative for a scalable and functional platform.[24]

Some examples of IXI Modules are:

- **Masked Authenticated Messaging (MAM):** a data communication protocol which adds functionality to emit and access an encrypted data stream. It is used in an experimental project called Data Marketplace.
- **Hub IXI:** it takes care of some of the complexity related to input management for exchanges, wallets, faucets and other services. It should lower the barrier for exchanges to list IOTA on their platform.
- **Flash Channels:** a payment channel to exchange instant off-Tangle transactions.

The last of the aforementioned IXI Modules is the subject of the next chapter.

3 Flash Channels Theory and Origins

At the moment IOTA and IoT is in a early phase of what is considered to be a transition period toward a production ready deployment and IoT environment.

Until the network grows considerably, and beyond that the IoT landscape starts adopting DLT natively with hardware support, which is supposed to give IOTA unlimited throughput for all practical purposes, transactions can't be really instant.

For use cases which require very rapid high throughput of transactions IOTA developed Flash Channels.

They are not a completely new idea because they belong to the well-known concept of payment channel.

Other cyptocurrencies such as Bitcoin and Ethereum have developed similar initiatives, called respectively Lightning Networks and Raiden.

In the next section we will explain what is a payment channel and we will try to make a comparison between Lightning Networks and Flash Channels.

3.1 State and Payment Channels

State channel represents an off-chain alteration of state, secured by eventual settlement in a DLT.

A payment channel is a state channel where the state being altered is the balance of a virtual currency.

Payment channels are a trustless mechanism for exchanging cryptocurrency transactions between two or more parties, outside of the DLT. These transactions, which would be valid if settled on the DLT, are held off-chain instead, acting as promissory notes for eventual batch settlement. Because the transactions are not settled, they can be exchanged without the usual settlement latency, allowing extremely high transaction throughput, low (submillisecond) latency, and fine granularity.

Actually, the term channel is a metaphor. State channels are virtual constructs represented by the exchange of state between two or more parties, outside of the DLT. There are no "channels" per se and the underlying data transport mechanism is not the channel. We use the term channel to represent the relationship and shared state between two or more parties, outside of the DLT.

To further explain this concept, think of a TCP stream. From the perspective of higher-level protocols it is a "socket" connecting two applications across the internet. But if we look at the network traffic, a TCP stream is just a virtual channel over IP packets. Each endpoint of the TCP stream sequences and assembles IP packets to create the illusion of a stream of bytes. Underneath, it's all disconnected packets. Similarly, a payment channel is just a series of transactions. If properly sequenced and connected, they create redeemable obligations that we can trust even though we don't trust the other side of the channel.[4]

Basic concepts and terminology A state channel is established between two or more parties, through a transaction that locks a shared state on the DLT. This is called the *funding transaction* or anchor transaction. This single transaction must

be transmitted to the network and mined (in case of blockchain) to establish the channel. In the example of a payment channel, the locked state is the initial balance (in currency) of the channel.

The two parties then exchange signed transactions, called *commitment transactions*, that alter the initial state. These transactions are valid transactions in that they could be submitted for settlement by either party, but instead are held off-chain by each party pending the channel closure. State updates can be created as fast as each party can create, sign, and transmit a transaction to the other party. In practice this means that thousands of transactions per second can be exchanged.

When exchanging commitment transactions the two parties also invalidate the previous states, so that the most up-to-date commitment transaction is always the only one that can be redeemed. This prevents either party from cheating by unilaterally closing the channel with an expired prior state that is more favorable to them than the current state. We will examine the various mechanisms that can be used to invalidate prior state in the rest of this section.

Finally, the channel can be closed either *cooperatively*, by submitting a final *settlement transaction* to the blockchain, or *unilaterally*, by either party submitting the last commitment transaction to the DLT. A unilateral close option is needed in case one of the parties unexpectedly disconnects. The settlement transaction represents the final state of the channel and is settled on the DLT.

In the entire lifetime of the channel, only two transactions need to be published on the DLT: the funding and settlement transactions. In between these two states, the two parties can exchange any number of commitment transactions that are never seen by anyone else, nor submitted to the DLT.

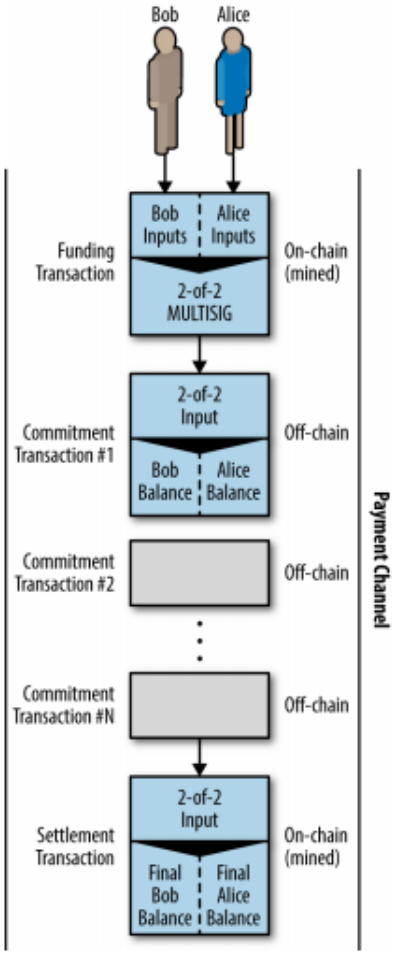

Figure 18: A payment channel between Bob and Alice, showing the funding, commitment and settlement transactions

Simple payment channel To explain state channels, we have to start with a very simple example. In this and the subsequent examples we will refer to Bitcoin and the blockchain, but the basic concepts can be applied to all kinds of payment channel.

We demonstrate a one-way channel, meaning that value is flowing in one direction only. We will also start with the naive assumption that no one is trying to cheat, to keep things simple.

Once we have the basic channel idea explained, we will then look at what it takes to make it trustless so that neither party can cheat, even if they are trying to.

For this example we will assume two participants: Emma and Fabian. Fabian offers a video streaming service that is billed by the second using a micropayment channel. Fabian charges 0.01 millibit (0.00001 BTC) per second of video, equivalent to 36 millibits (0.036 BTC) per hour of video. Emma is a user who purchases this streaming video service from Fabian.

To set up the payment channel, Emma and Fabian establish a 2-of-2 multisignature address, with each of them holding one of the keys. The multisig address is then funded by Emma. Emma's transaction, paying to the multisignature address,

is the funding or anchor transaction for the payment channel.

For this example, let's say that Emma funds the channel with 36 millibits (0.036 BTC). This will allow Emma to consume up to 1 hour of streaming video. The funding transaction in this case sets the maximum amount that can be transmitted in this channel, setting the channel capacity.

The funding transaction consumes one or more inputs from Emma's wallet, sourcing the funds. It creates one output with a value of 36 millibits paid to the multisignature 2-of-2 address controlled jointly between Emma and Fabian. It may have additional outputs for change back to Emma's wallet.

Once the funding transaction is confirmed, Emma can start streaming video.

Emma creates and signs a commitment transaction that consumes the 36 millibits output created by the funding transaction and creates two outputs: one for her refund, the other for Fabian's payment.

The transaction is spending tokens from a multisig address, therefore it needs to be signed by Fabian as well to be considered valid.

When Fabian receives this transaction, he adds the second signature and returns it to Emma together with 1 second worth of video.

Now both parties have a fully signed commitment transaction that either can redeem, representing the correct up-to-date balance of the channel. Neither party broadcasts this transaction to the network.

In the next round, Emma creates and signs another commitment transaction that consumes the same 2-of-2 output from the funding transaction. The second commitment transaction allocates one output of 0.2 millibits to Fabian's address and one output of 35.98 millibits back to Emma's address.

This new transaction is payment for two cumulative seconds of video. Fabian signs and returns the second commitment transaction, together with another second of video.

In this way, Emma continues to send commitment transactions to Fabian in exchange for streaming video. The balance of the channel gradually accumulates in favor of Fabian, as Emma consumes more seconds of video.

Finally, Emma clicks "Stop" to stop streaming video. Either Fabian or Emma can now transmit the final state transaction for settlement. This last transaction is the settlement transaction and pays Fabian for all the video Emma consumed, refunding the remainder of the funding transaction to Emma.

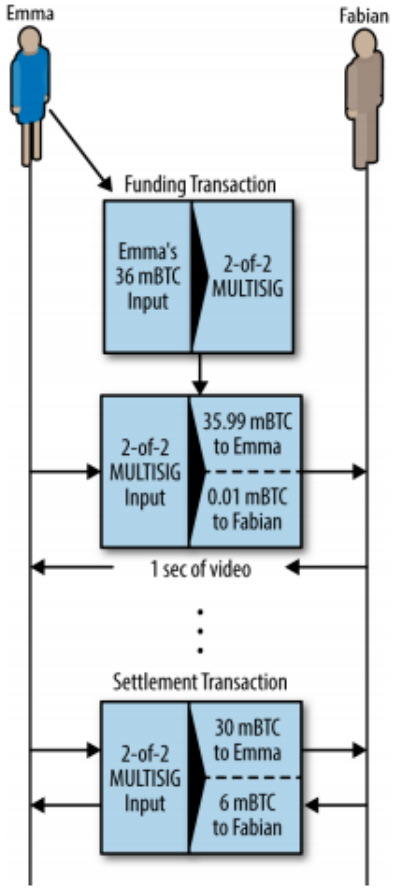

Figure 19: A payment channel between Bob and Alice. Each blue box is a transaction. On the left side there are input, on the right side there are outputs

Trustless channels: one first possible solution The channel we just described works, but only if both parties cooperate, without any failures or attempts to cheat.

Anyway cryptocurrency applications are meant to work in a trustless environment, therefore we have to take into account some possible cheating scenario

- Once the funding transaction happens, Emma needs Fabian's signature to get any money back. If Fabian disappears, Emma's funds will remain locked in the multisig and effectively lost forever. This channel, as constructed, leads to a loss of funds if one of the parties disconnects before there is at least one commitment transaction signed by both parties
- While the channel is running, Emma can take any of the commitment transactions Fabian has countersigned and transmit one to the blockchain. Emma could decide to put in the blockchain not the last transaction, but the one most favorable to her. Why pay for 600 seconds of video, if she can transmit the first commitment transaction and only pay for 1 second of video? The channel fails because Emma can cheat by broadcasting a prior commitment that is in her favor.

To solve this problems, Bitcoin uses one of its most important feautures: Timelocks.

A *Timelock* is a type of smart contract primitive that restricts the spending of some bitcoins until a specified future time or block height.

Bitcoin has had a transaction-level timelock feature from the beginning. It is implemented by the *nLocktime* field in a transaction.

It is set to zero in most transactions to indicate immediate propagation and execution. If nLocktime is nonzero and below 500 million, it is interpreted as a block height, meaning the transaction is not valid and is not relayed or included in the blockchain prior to the specified block height. If it is above 500 million, it is interpreted as a Unix Epoch timestamp (seconds since Jan-1-1970) and the transaction is not valid prior to the specified time.

Transactions with nLocktime specifying a future block or time must be held by the originating system and transmitted to the bitcoin network only after they become valid.

If a transaction is transmitted to the network before the specified nLocktime, the transaction will be rejected by the first node as invalid and will not be relayed to other nodes. The use of nLocktime is equivalent to postdating a paper check.

In the previous example Emma cannot risk funding a 2-of-2 multisig unless she has a guaranteed refund.

A refund is a transaction that takes as input all the tokens in the multisig address and give them back to Emma.

To solve this problem, Emma constructs the funding and refund transaction at the same time. She signs the funding transaction but doesn't transmit it to anyone. Emma transmits only the refund transaction to Fabian and obtains his signature.

The refund transaction acts as the first commitment transaction and its timelock establishes the upper bound for the channel's life. In this case, Emma could set the nLocktime to 30 days or 4320 blocks into the future. All subsequent commitment transactions must have a shorter timelock, so that they can be redeemed before the refund transaction.

Now that Emma has a fully signed refund transaction, she can confidently transmit the signed funding transaction knowing that she can eventually, after the timelock expires, redeem the refund transaction even if Fabian disappears.

Every commitment transaction the parties exchange during the life of the channel will be timelocked into the future. But the delay will be slightly shorter for each commitment so the most recent commitment can be redeemed before the prior commitment it invalidates. Because of the nLockTime, neither party can successfully propagate any of the commitment transactions until their timelock expires.

The channel can be closed in two ways: bilaterally or unilaterally.

In the first case all the parties will cooperate and close the channel gracefully with a settlement transaction, making it unnecessary to transmit an intermediate commitment transaction.

In the second case the most recent commitment transaction can be propagated to settle the account and invalidate all prior commitment transactions.

For example, if commitment transaction #1 is timelocked to 4320 blocks in the future, then commitment transaction #2 is timelocked to 4319 blocks in the future ans so on.

Commitment transaction #600 can be spent 600 blocks before commitment

transaction #1 becomes valid.

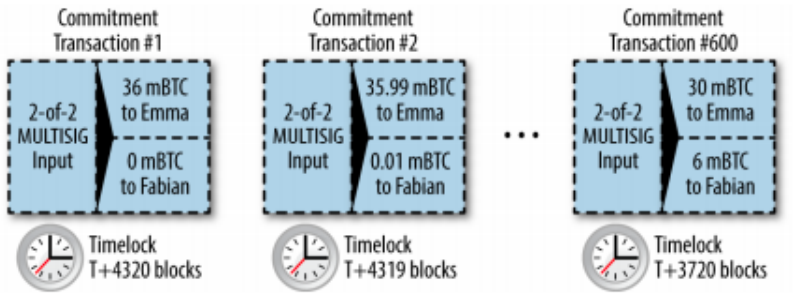

Figure 20: Each commitment sets a shorter timelock, allowing it to be spent before the previous commitments become valid

Each subsequent commitment transaction must have a shorter timelock so that it may be broadcast before its predecessors and before the refund transaction.

The ability to broadcast a commitment earlier ensures it will be able to spend the funding output and preclude any other commitment transaction from being redeemed by spending the output.

The guarantees offered by the bitcoin blockchain, preventing double-spends and enforcing timelocks, effectively allow each commitment transaction to invalidate its predecessors.

All commitment transactions are timelocked, therefore broadcasting a commitment transaction will always involve waiting until the timelock has expired.

But if the two parties agree on what the final balance is and know they both hold commitment transactions that will eventually make that balance a reality, they can decide to close the channel cooperatively constructing a settlement transaction without a timelock representing that same balance.

In this case either party takes the most recent commitment transaction and builds a settlement transaction that is identical in every way except that it omits the timelock. Both parties can sign this settlement transaction knowing there is no way to cheat and get a more favorable balance.

This allows both parties to redeem their balance immediately.

Timelocks are effective but they have two distinct disadvantages:

- *Limited channel lifetime*: by establishing a maximum timelock when the channel is first opened, they limit the lifetime of the channel. Worse, they force channel implementations to strike a balance between allowing long-lived channels and forcing one of the participants to wait a very long time for a refund in case of premature closure. For example, if you allow the channel to remain open for 30 days, by setting the refund timelock to 30 days, if one of the parties disappears immediately the other party must wait 30 days for a refund. The more distant the endpoint, the more distant the refund.

- *Limited number of offline transactions*: each subsequent commitment transaction must decrement the timelock, there is an explicit limit on the number of commitment transactions that can be exchanged between the parties. For example, a 30-day channel, setting a timelock of 4320 blocks into the future, can

only accommodate 4320 intermediate commitment transactions before it must be closed. There is a danger in setting the timelock commitment transaction interval at 1 block. By setting the timelock interval between commitment transactions to 1 block, a developer is creating a very high burden for the channel participants who have to be vigilant, remain online and watching, and be ready to transmit the right commitment transaction at anytime.

Trustless channels: a more flexible solution A better way to handle the prior commitment states is to explicitly revoke them.

However, this is not easy to achieve. A key characteristic of bitcoin is that once a transaction is valid, it remains valid and does not expire.

So instead of active revocation enforced by the blockchain, it's necessary to have a sort of smart contract that must be forcing both parties to broadcast only the most recent transaction. Any broadcast of older transactions will cause a violation of the contract, and all funds will be given to the other party as a penalty.

The way we do that is by giving each party a *revocation key* that can be used to punish the other party if they try to cheat. This mechanism for revoking prior commitment transactions was first proposed as part of the Lightning Network.

To explain how it works, let's suppose two users, Hitesh and Irene start the channel by collaboratively constructing a funding transaction, each funding the channel with 5 bitcoin. The initial balance is 5 bitcoin for Hitesh and 5 bitcoin for Irene.

The funding transaction locks the channel state in a 2-of-2 multisig, just like in the previous example.

The difference is in the construction of commitment transactions: instead of creating a single commitment transaction that both parties sign, Hitesh and Irene create two different commitment transactions that are asymmetric.

Before creating the commitment transactions, each party creates a revocation key that is kept secret.

After that, Irene creates a commitment transaction with two outputs. The first output pays Hitesh the 5 bitcoin he is owed immediately. The second output is a script containing a delayed output.

The script for that output allows Irene to redeem it after 1000 blocks *or* Hitesh to redeem it immediately if he has a revocation key for that transaction.

The transaction outputs look like this:

Input: 2-of-2 funding output, signed by Irene

Output 0 <5 bitcoin>:
    <Irene's Public Key> CHECKSIG

Output 1:
    <1000 blocks>
    CHECKSEQUENCEVERIFY
    DROP

<Hitesh's Public Key> CHECKSIG

Irene signs this transaction and sends it to Hitesh.
In the meanwhile Hitesh creates a specular transaction that looks like this:

```
 Input:  2-of-2 funding output, signed by Hitesh

Output 0 <5 bitcoin>:
   <Hitesh's Public Key> CHECKSIG

Output 1:
   <1000 blocks>
   CHECKSEQUENCEVERIFY
   DROP
   <Irene's Public Key> CHECKSIG
```

Hitesh signs this transaction and sends it to Irene.

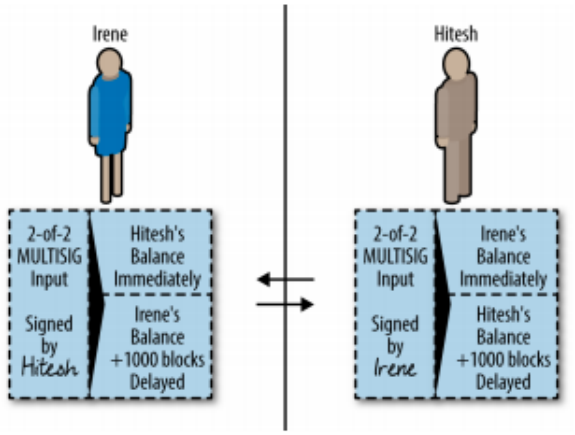

Figure 21: Hitesh and Irene exchange asymmetric commitment transactions

Now both of them have transaction signed by the other party. By adding the missing signature to those transaction, any of them can publish one of these transaction to the blockchain.

Anyway both of them has little incentive to do so, because they would have to wait 1000 blocks before being able to redeem their part of output.

By imposing a delay on the redemption of one of the outputs, we put each party at a slight disadvantage when they choose to unilaterally broadcast a commitment transaction. But a time delay alone isn't enough to encourage fair conduct.

Now we introduce the final element of this scheme: a revocation key that allows a wronged party to punish a cheater by taking the entire balance of the channel.

Each of the commitment transactions has a "delayed" output. The redemption script for that output allows one party to redeem it after 1000 blocks or the other party to redeem it if they have a revocation key.

So when Hitesh creates a commitment transaction for Irene to sign, he makes the second output payable to himself after 1000 blocks, or to whoever can present a revocation key. Hitesh constructs this transaction and creates a revocation key that he keeps secret. He will only reveal it to Irene when he is ready to move to a new channel state and wants to revoke this commitment. The script looks like this:

```
 Input:  2-of-2 funding output, signed by Hitesh

Output 0 <5 bitcoin>:
   <Irene's Public Key> CHECKSIG

Output 1:
   IF
      # Revocation penalty output
      <Revocation Public Key>
   ELSE
      <1000 blocks>
      CHECKSEQUENCEVERIFY
      DROP
      <Hitesh's Public Key> CHECKSIG
   ENDIF
   CHECKSIG
```

Irene can confidently sign this transaction, since if transmitted it will immediately pay her what she is owed. Once Irene has signed the transaction and has sent it back to Hitesh, he can transmit the transaction as well, but knows that if he transmits it in a unilateral channel closing, he will have to wait 1000 blocks to get paid.

When the channel is advanced to the next state, Hitesh has to revoke this commitment transaction before Irene agrees to sign the next commitment transaction. To do that, all he has to do is send the revocation key to Irene. Once Irene has the revocation key for this commitment, she can sign the next commitment with confidence. She knows that if Hitesh tries to cheat by publishing the prior commitment, she can use the revocation key to redeem Hitesh's delayed output. If Hitesh cheats, Irene gets *both* outputs.

This is possible because `Output 1` is spendable by whoever owns the revocation key. If both parties have such a key, both of them can publish in the blockchain a transaction spending that output.

If Hitesh tries to cheat by issuing the old commitment transaction, he can - immediately after the transaction has been published in a block by some miner - issue a new transaction spending `Output 1`. Anyway Irene can do the same thing as well! The two transactions would be in conflict, therefore just one of them will be confirmed eventually.

The revocation protocol is bilateral, meaning that in each round, as the channel state is advanced, the two parties exchange new commitments, exchange revocation keys for the previous commitment, and sign each other's commitment transactions.

As they accept a new state, they make the prior state impossible to use, by giving each other the necessary revocation keys to punish any cheating.

Importantly, the revocation doesn't happen automatically. While one party has the ability to punish the other one for cheating, it has to watch the blockchain diligently for signs of cheating. If it sees a prior commitment transaction broadcast, it has 1000 blocks to take action and use the revocation key to thwart the cheating and punish the other party by taking the entire balance.

Asymmetric revocable commitments with relative time locks (CSV) are a much better way to implement payment channels and a very significant innovation in this technology. With this construct, the channel can remain open indefinitely and can have billions of intermediate commitment transactions. In prototype implementations of Lightning Network, the commitment state is identified by a 48-bit index, allowing more than 281 trillion (2.8 x 1014) state transitions in any single channel!

Hash Time Lock Contracts (HTLC) Before explaining how Lightning Networks work, we introduce one last concept that is extensively used in their implementation.

This feature is called Hash Time Lock Contract, or HTLC, and is a special type of smart contract that allows the participants to create payments with a certain expiration date.

A HTLC contract consists of two parts: hash verification and time expiration verification.

Let us start with the hash. To create an HTLC, the intended recipient of the payment will first create a secret R. They then calculate the hash of this secret $H = Hash(R)$.

This produces a hash H that can be included in an output's locking script. Whoever knows the secret can use it to redeem the output. The secret R is also referred to as a *preimage* to the hash function. The preimage is just the data that is used as input to a hash function.

The second part of the HTLC contract is the verification of the expiration time of the payment. If the secret was not revealed in time and the payment was not used, the sender can retrieve all the funds.

The script implementing an HTLC might look like this:

```
 IF
   # Pay who presents the secret R
   HASH160 <H> EQUALVERIFY
ELSE
   # Refund the payer after timeout
   <locktime> CHECKLOCKTIMEVERIFY DROP
   <Payee Pubic Key> CHECKSIG
ENDIF
```

Anyone who knows the secret R, which when hashed equals to H, can redeem this output by exercising the first clause of the IF flow.

If the secret is not revealed and the HTLC claimed, after a certain number of blocks the payer can claim a refund using the second clause in the IF flow.

This is a basic implementation of an HTLC. This type of HTLC can be redeemed by anyone who has the secret R.

An HTLC can take many different forms with slight variations to the script. For example, adding a CHECKSIG operator and a public key in the first clause restricts redemption of the hash to a named recipient, who must also know the secret R.

3.2 Lightning Networks

Now that we finally have analyzed all the components, we can consider the entire picture of how the Lightning Network works.

The Lightning Network is a proposed routed network of bidirectional payment channels connected end-to-end. A network like this can allow any participant to route a payment from channel to channel without trusting any of the intermediaries. The Lightning Network was first described by Joseph Poon and Thadeus Dryja in February 2015, building on the concept of payment channels as proposed and elaborated upon by many others.

"Lightning Network" refers to a specific design for a routed payment channel network, which has now been implemented by at least five different open source teams. The independent implementations are coordinated by a set of interoperability standards described in the Basics of Lightning Technology (BOLT) paper.

Prototype implementations of the Lightning Network have been released by several teams. For now, these implementations can only be run on testnet because they use segwit, which is not activated on the main bitcoin blockchain (mainnet).

The Lightning Network is one possible way of implementing routed payment channels. There are several other designs that aim to achieve similar goals, such as Teechan and Tumblebit.

To better understand Lightning Networks working principles, let's see an example.

We have five participants: Alice, Bob, Carol, Diana, and Eric.

These five participants have opened payment channels with each other, in pairs. Alice has a payment channel with Bob. Bob is connected to Carol, Carol to Diana, and Diana to Eric. For simplicity let's assume each channel is funded with 2 bitcoin by each participant, for a total capacity of 4 bitcoin in each channel.

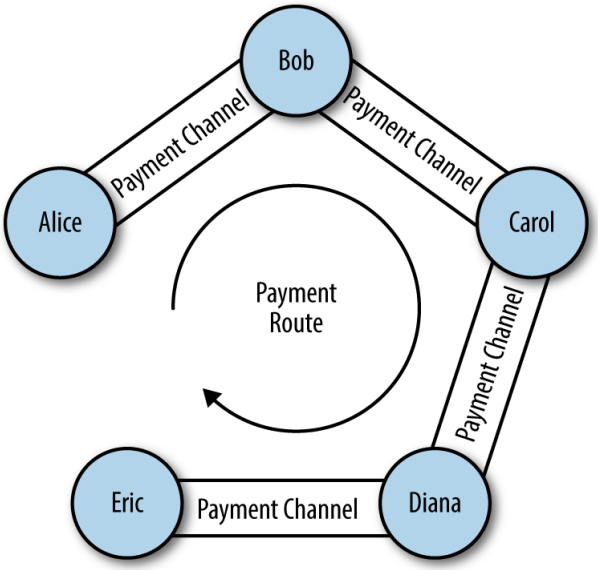

Figure 22: A series of bidirectional payment channels linked to form a Lightning Network that can route a payment from Alice to Eric

Let us assume that Alice wants to transfer 1 bitcoin to Eric. However, as we see, they are not connected by a direct channel, and opening a new channel requires time and money (the funding transaction to be put in the blockchain, like any other transaction, has a fee to be paid to the miner). Luckily, Alice is connected to the Lightning Network and can make an indirect payment with the help of a series of HTLC contracts. Let us examine this payment step by step.

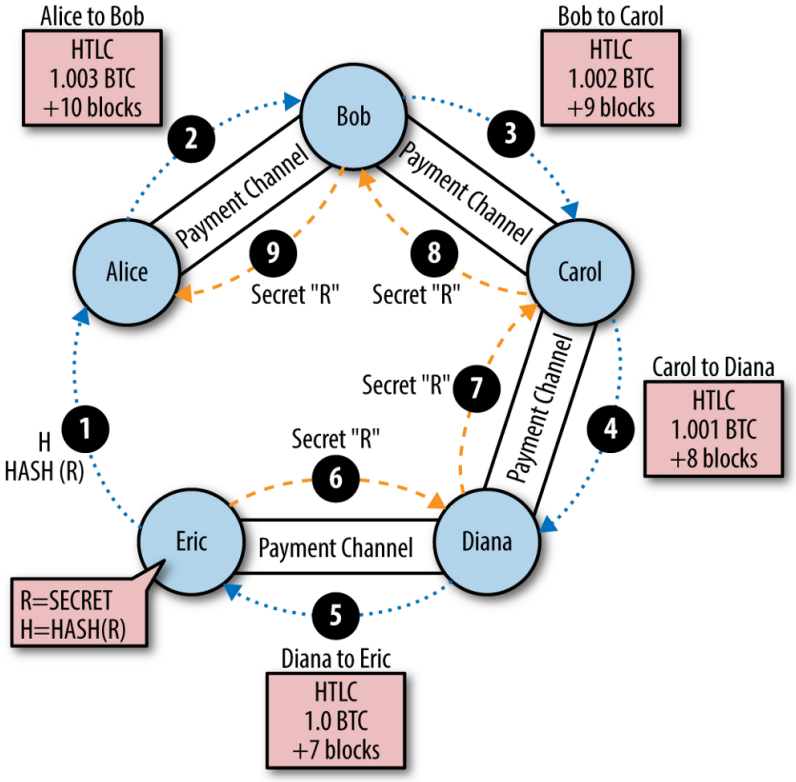

Figure 23: Step-by-step payment routing through a Lightning Network

1. Eric creates a secret R and communicates its hash to Alice (he does not show the secret itself to anyone).

2. Alice uses this hash to create a HTLC with a timelock set on 10 blocks forward, for the amount of 1.003 BTC. The additional 0.003 BTC will be used as a fee to the intermediate participants for their participation in the chain. So, Alice locks her balance of 1.003 BTC in the HTLC, using the following rule: *"Alice will pay Bob 1.003 BTC if he finds the secret R during the next 10 blocks, otherwise the funds will return to Alice"*. The balance of the channel is now expressed by a commitment transaction with three outputs: 2 BTC to Bob, 0.997 BTC to Alice, and 1.003 BTC locked in Alice's HTLC. Alice's balance is reduced by the amount committed to the HTLC.

3. In his turn, Bob, having at his disposal the Alice's commitment transaction (the HTLC sent to the channel that connects them), creates a HTLC on the channel that connects him to Carol for the amount of 1.002 BTC with a timelock set on 9 blocks forward.
   He uses the same hash as Alice. Bob knows that Carol will have to find out the secret R to unlock the HTLC sent by him, and as soon as she does it, he will also learn it and will therefore be capable to unlock the 1.003 BTC sent to him by Alice. If Carol cannot find out the secret R, Bob and Alice will simply retrieve their funds on the expiration of the timelock.
   Let us also remark that the amount of funds sent by Bob is 0.001 BTC smaller than the funds he receives: it is a fee he took for his participation in the chain.

The balance of the channel between Bob and Carol is as following: Carol has 2 BTC, Bob has 0.998 BTC, and 1.002 BTC are locked in the HTLC.

4. Carol, having received Bob's commitment transaction, creates an HTLC (using the hash that Bob used) on the channel with Diana for the amount of 1.001 BTC with a timelock set on 8 blocks forward.
   If Diana can discover the secret R before the 8 blocks are over and unblock the HTLC to get 1.001 BTC, Carol will also learn the secret and will therefore be able to unlock the 1.002 BTC sent to her by Bob, therefore earning 0.001 BTC.
   The balance of the channel between Carol and Diana is now as following: Diana has 2 BTC, Carol has 0.999 BTC, and 1.001 BTC are locked in the HTLC.

5. Finally, Diana sends a HTLC (always using the same hash) to her channel with Eric for the amount of 1 BTC and a timelock set on 7 blocks forward.
   The balance of the channel between Diana and Eric is now as following: Eric has 2 BTC, Diana has 1 BTC, and 1 BTC are locked in the HTLC.

6. Now, we have arrived to the end of our chain. Eric, having the secret R, whose hash was used in all the HTLC commitment transactions, can unlock the HTLC sent to him by Diana and obtain 1 BTC. As soon as Eric uses the secret to receive the funds, it becomes available to Diana as well.
   The balance of the channel between Diana and Eric is now as following: Eric has 3 BTC and Diana has 1 BTC.

7. Diana, having received the secret, unlocks 1.001 BTC sent by Carol and, by doing so, reveals her the secret. The balance of their channel is now as following: Diana has 3.001 BTC and Carol has 0.999 BTC.

8. Carol, having received the secret, unlocks 1.002 BTC sent by Bob and, by doing so, reveals him the secret. The balance of their channel is now as following: Carol has 3.002 BTC and Bob has 0.998.

9. Finally, Bob uses the secret to obtain 1.003 BTC from the channel between him and Alice. The balance of their channel is now as following: Bob has 3.003 BTC and Alice has 0.997.

   In this way, Alice paid Eric 1 BTC without opening a new channel that would link them directly. No one of the chain participants was obliged to trust others, and they earned 0.001 BTC as a fee for their role as middlemen.

In the case of anybody from the chain failing to perform their part of work, nobody would have lost the money that would simply stay frozen until the expiration of the timelock.

Evidently, the longer the chain used to deliver the funds, the higher the probability that they would not reach the end: some of the participants may close the channel unilaterally or simply lose the Internet connection. Let us examine two possibilities of something going wrong.

Broken Channel In the first case, let us imagine that the funds have successfully reached the end of the chain, but on the way back (when the secret should go back in the chain, reaching the very beginning) a problem has emerged when one participant refused or was unable to cooperate. In our example, it would be Bob.

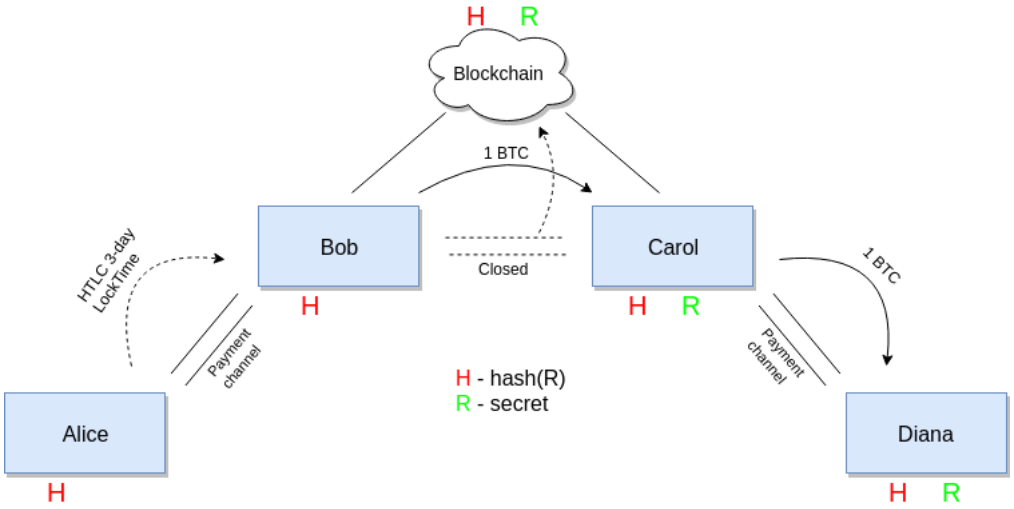

Figure 24: Failure in funds delivery due to one of the channels closing

Diana, when the chain has reached her, immediately retrieves her funds, using the secret and revealing it to Carol.

Carol also wishes to get her money back from Bob but he does not respond so, to avoid risk, she closes the channel with him, sending the last commitment transaction (the HTLC contract previously sent by Bob) into the blockchain.

This transaction assigns 0.998 BTC to Bob, 2 BTC to Carol and 1.002 BTC stay locked in Bob's HTLC.

Using the secret Carol can redeem the locked 1.002 BTC.

In this case, Bob still has three days to surface and took his money from Alice (as the transaction has made it to the blockchain, he can easily find it and see the R). Otherwise, at the expiration of the timelock, Alice will be able to retrieve all her money.

In this second case Bob has lost money. Let's see the net balance in the two channels he is involved in:

- Channel Alice-Bob: at the opening he funded the channel with 2 BTC. At the closure he redeems 2 BTC
- Channel Bob-Carol: at the opening he funded the channel with 2 BTC. At the closure he redeems 0.998 BTC.

It can be seen that if a participant for some reason leaves the system, he or she is the only one who risks losing funds, while all other participants of the chain are safe.

Rerouting In our second case, let us examine a situation in which the funds did not reach the end of the chain, again because of a problem with one of the participants. Now it would be Carol.

The first and the most obvious way to handle this scenario is simply to wait until the expiration time of the HTLC contracts to be able to retrieve the funds and send a new payment.

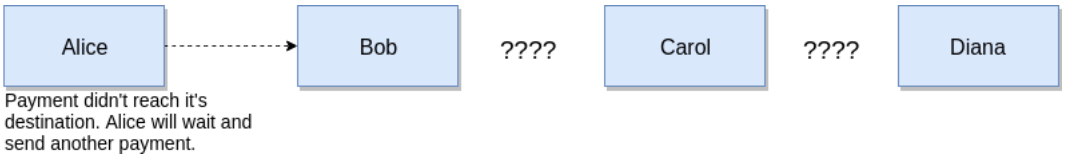

Figure 25: One of the nodes in the payment route is unresponsive

But what should they do if Alice is in a hurry? Of course, it is possible to send a new payment by another chain without waiting for the return of the funds, but what if Carol comes back, connects with Bob and finishes the chain? In this case, Alice would end sending the double amount of money.

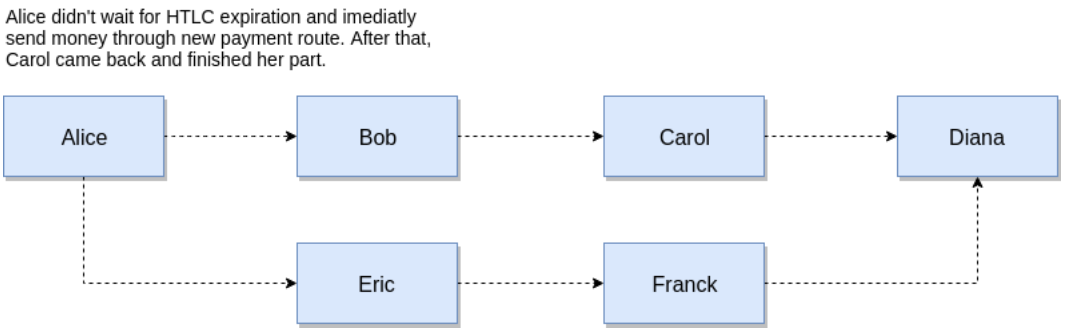

Figure 26: Alice have built another payment route

So should we always, in the case of failure, wait until the expiration time to send a new payment? Fortunately, to avoid waiting for the time lock to expire, we may cancel the previous payment.

To do this, Diana (the recipient) should send an equivalent amount of money to Alice, using the same hash as when sending it for the first time, and the chain can be different. Now, if Carol comes back and finishes her part of the work, the money will simply go around the circle. It means that the failed payment can be considered null and another payment can be sent by another chain.

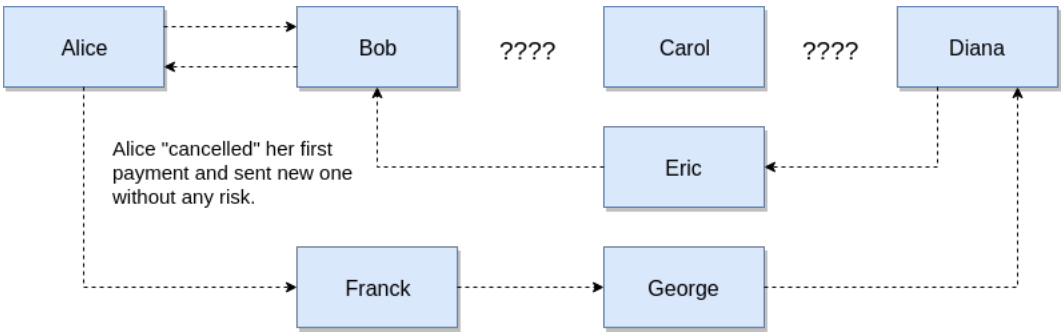

Figure 27: Alice canceled an old payment and now can send new one without any risk

Main benefits

- **Scalability**: only two transaction have to be published in the blockchain during the channel lifetime. All commitment transactions are held off-chain so they don't overload the network allowing to considerably increase the TPS rate of Bitcoin.
- **Instant transactions**: commitment transactions don't have to be mined or confirmed, so the only bottleneck is the time it takes to transmit them from the sender to the receiver
- **Micropayments**: commitment transactions are exchanged off-chain, so no fee has to be paid for them. This enables use cases where the the amount of money to be sent is minimal and frequent, such as pay-per-use policies in video streaming services
- **Privacy**: transactions are much more anonymous than in the bitcoin blockchain because the participants of the chains used to make a payment cannot see the sender or the recipient.

3.3 Flash Channels

IOTA, unlike Bitcoin, has no problems of scalability and, if the network will keep growing, is supposed to be able reach a very high confirmed transaction rate (CTPS). Transactions will be confirmed very quickly but even in the best case scenario they will never be instant.

Indeed PoW has to be performed for each transaction. At the moment it takes few minutes on an average PC. Maybe in the future it will be performed using ad-hoc hardware (Jinn trinary chip is being developed by IOTA stakeholders) and it will only require few seconds to be computed.

Anyway PoW in IOTA exists mainly as anti-spam measure, therefore it will always have to take some not negligible time to be effective.

After transaction creation, some time will be necessary for broadcasting it to a sufficient number of nodes and further time for others transactions to confirm the issued transaction.

At the time of writing confirmation takes at least some minute, but in many cases much more time and in some unfortunate case valid transactions are never going to be confirmed, making it necessary to rebroadcast the transaction.

These motivations are good reasons to believe that despite the clear advantages over blockchain, the Tangle will never be able to enable really instant and high throughput transactions.

That's why IOTA Foundation decided to develop an IXI Module to support a payment channel with similar objective as Lightning Networks in Bitcoin.

The name of this IXI Module is Flash Channels.

Multisig Flash Channels heavily use addresses signed by more than one party, called *multisig addresses*. Of course this concept also exists in Bitcoin, but to better understand how Flash works, it is useful to make a brief introduction on how multisig works in IOTA.[19]

Multi-signature is a digital signature scheme which allows a group of users to sign a single document.

This group of users is often referred to as "co-signers" and has to formally agree in order to spend money from a wallet. As such, for each co-signer, the only way to successfully spend tokens is if the signature of the other co-signers is provided as well.

There are various ways to use multi-signatures: there can be N-of-N schemes, meaning that all co-signers need to sign for a transfer to be successful or M-of-N, meaning that only a subset of all co-signers needs to provide their signatures in order to make a transaction.

This makes multi-signature a perfect solution to hold tokens in a trustless and secure manner.

To understand the difference between a normal address and a multisig address let's see how a normal address is generated.

To generate a normal address, the starting point is a private key, that is generated from a combination of seed and a index. With a complex sequence of repeated hashings, from a private key is obtained a so called *digest*.

By hashing a digest once we obtain an address.

Addresses can have different level of security but every address has 81 trytes length, regardless of the security level.

However, in case a higher level is selected, private key and digest are of different length, and length depends on the level of security.

For example, if security=10, private key is 21870(=2187*10) trytes, digest is 810(=81*10) trytes but address is 81 trytes.

Multisig makes use of this feature that the final generated address is always of 81 trytes long, no matter how long the original digest was.

For instance, let's say Alice and Bob want to create an address both of them are responsible of. They have their own unique seed and neither of them must know the other's.

First of all, they generate the address called multisig address, which looks like a normal address of 81 trytes from any Tangle observer, but is generated in different way.

To generate a multisig address(2-of-2 in this case), two digests are required: one generated by Alice and the other one by Bob. For each of them, they can individually choose their own index and security level independently.

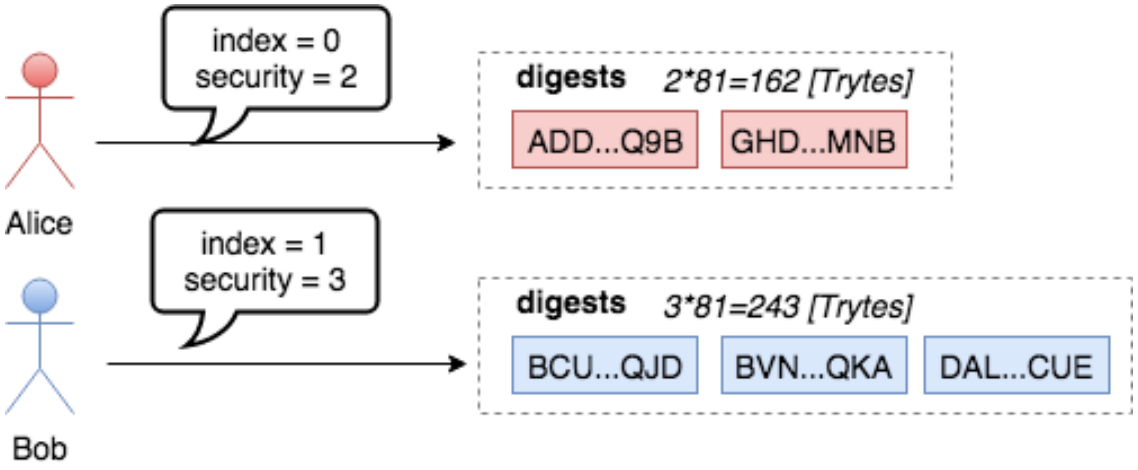

Figure 28: Alice and Bob generate their own digests independently

Note that during the whole process they never share their seeds. Indeed they can share their digests publicly because it's not possible to go back up to the seeds from them.

Next step is to concatenate in an agreed upon order these submitted two digests to get one longer digest.

In our example this new digest looks like a digest of security=5. Then using the same method used in normal address generation, the digest is hashed once to create an address. That final address is the so called multisg address and is always 81 trytes long, no matter how long the original digest was.

In summary, two different digests, each of them created from a different seed and concatenated together are seen as one digest. From this digest is generated the multisg address.

By concatenating more than two digests, it's possible to generate addresses that require more than two co-signers.

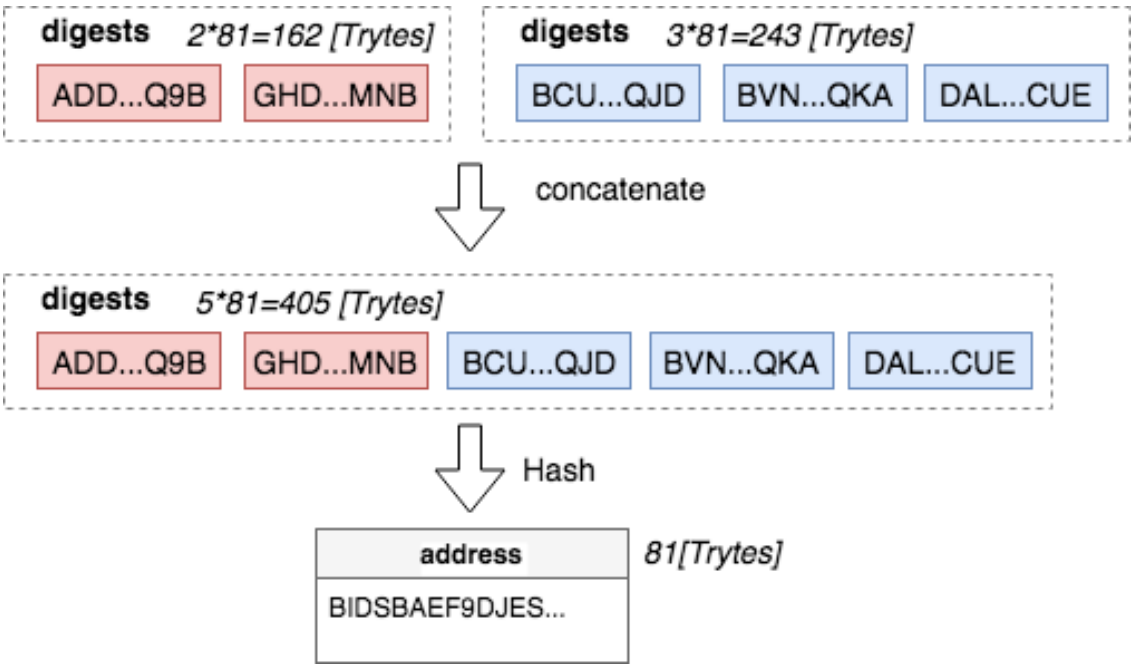

Figure 29: Alice's and Bob's digests are concatenated to obtain a longer digest. By hashing this digest a multisig address is obtained

Even if multisig addresses are created in a different way than ordinary addresses, when attached to the Tangle, they appear to be not different and people cannot distinguish multisig attached bundles from others.

As a consequence, sending money *to* a multisig address is exactly the same as sending money to a normal address.

The difference is in sending *from* a multisig address.

Spending from any address requires a signature made with the private key associated to that address.

In multisig case more than one private key is associated to the address, therefore more than one signature is required.

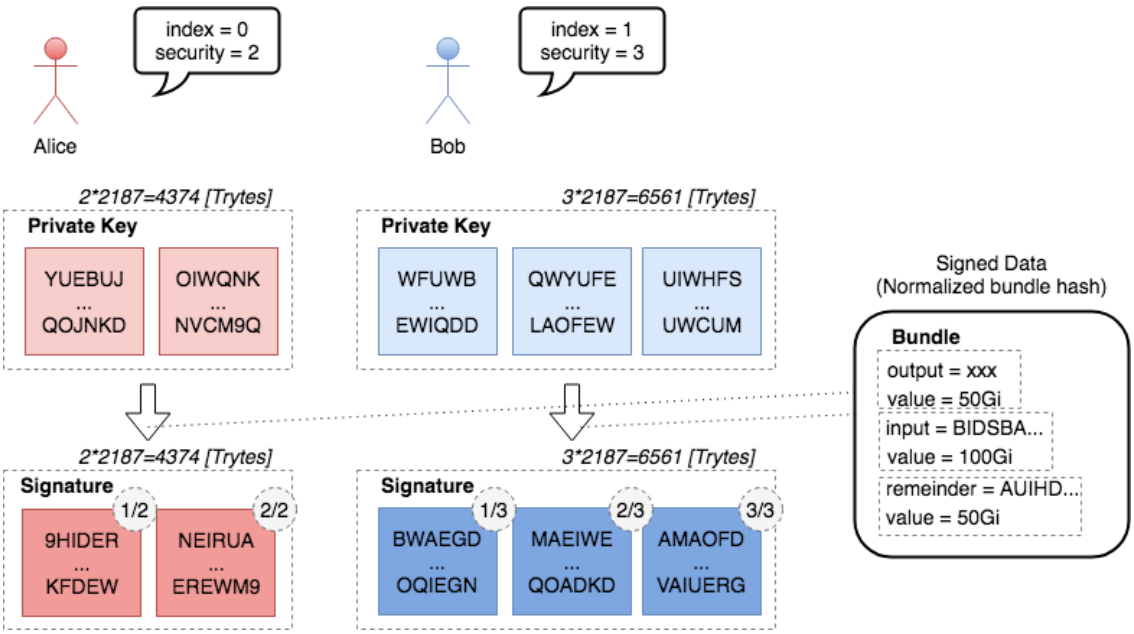

Figure 30: Alice and Bob sign the bundle with their own private key

Now we have co-signers signature, two signatures in total.

In this example, Alice's address has security=2 and Bob's has security=3, so Alice's signature is of $2187 * 2 = 4372$ trytes, and Bob's is $2187 * 3 = 6561$ trytes. In total $4372 + 6561 = 10935$ trytes are stored in the *signatureFragment* field of five transaction objects in the bundle (each *signatureFragment* is 2187 trytes, therefore $10935/2187 = 5$ fragments are required).

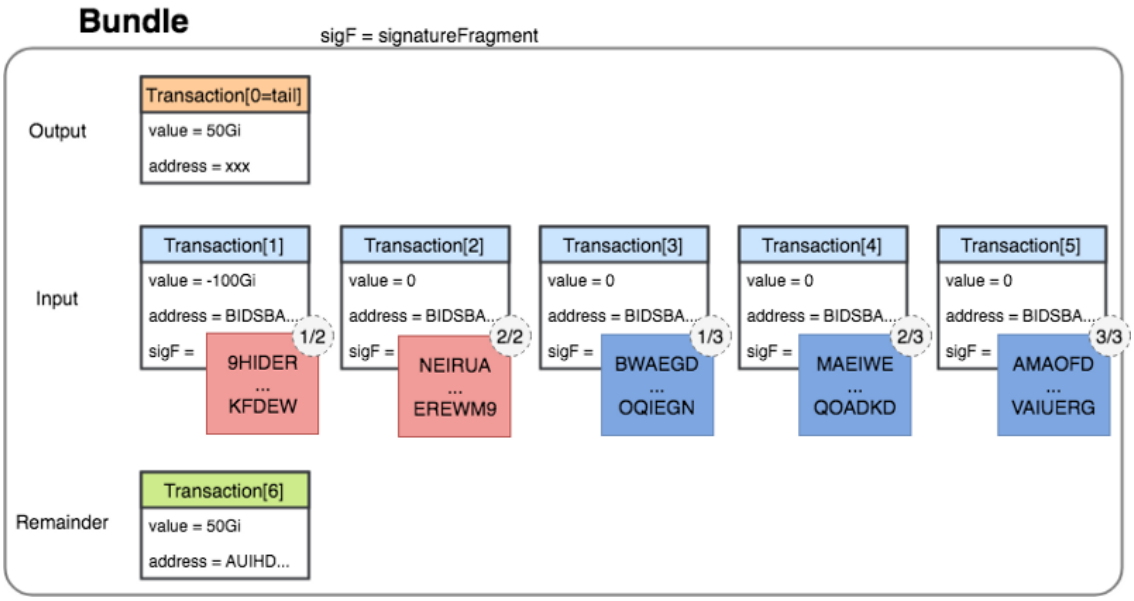

Figure 31: Overview of the final bundle signed by both parties

It's very important to remark that signatures must be provided in the same order as the co-signers added their key digest when they generated the multisig

address.

Validating is easy. On the Tangle, multisig bundles look just the same as the ones created from single signature.

The only differences are the length of the signature parts. As described in the case of single signature, a different security level creates a different length of the signature ($security * 2187$ trytes).

For example a 3-of-3 multisig bundle signed by 3 parties whose addresses have security=(3,3,2) can be validated as a single signature of security $3 + 3 + 2 = 8$.

When spending inputs, it's important to keep in mind what happens with the remainder.

If there is a remainder, the co-signers need to define a new multi-signature address to which the remainder balance gets sent to. This means that this new multi-signature address needs to be generated before making the transaction.

What is a Flash Channel Flash is a bi-directional off-Tangle payment channel to enable instantaneous, high throughput transactions. In essence, they provide a way for parties to transact at high frequency without waiting for each transaction to confirm on the public IOTA network. Instead, only two transactions will ever occur on the main IOTA network: opening and closing transactions of the Flash Channel.

An off-Tangle approach reduces the per-transaction overhead to a negligible level by creating signed transactions off the Tangle and opens up a feeless transaction model for instant token streaming.

On a practical level a Flash Channel is a data structure shared among all parties participating to the channel.

IOTA does not enforce a particular data structure representing the channel state, but there is a recommended one that will be analyzed in the next paragraphs. Developers are free to modify such a data structure, provided that it remains compliant with the official Flash Channels API library.

More importantly there are no rules on what communication protocol the parties should follow or what physical layer should be used to exchange off-Tangle transactions and data.

Using WiFi connection is an option, but it's not the only one. For example some PoC applications have been developed using Bluetooth as a standard for short range communication.

In general any means of communication is good for exchanging off-Tangle transaction. Internet connectivity is only required to issue the initial and final transaction to the Tangle.

Even in that case it should be possible to avoid the necessity of Internet connectivity by sending the transaction to a full node that will take care of finding trunk and branch transactions to reference, computing PoW and broadcast the transaction to the network.

As previously said, the purpose of Flash Channels is the same as Lightning Network for Bitcoin and Raiden for Ethereum. Anyway there are big differences among those cryptocurrencies due to the DLT they use as a backbone.

Therefore some major differences in the implementation has been made to make them secure and convenient in the trustless environment they must work on.

The main differences (both positive and negative) between IOTA and Bitcoin that had consequences on the differences in the implementation choices of Flash Channels over Lightning Networks are:

- IOTA does not enforce timestamps and has no smart contracts. This removes the possibility to use the timelock based and revocation key mechanisms that we have previously seen in Bitcoin to avoid cheating by one of the parties. Incentives to close the channel correctly must be provided in another way
- Since IOTA is feeless, opening and closing channels is free. Therefore Flash is not concerned with routing as Lightning Network
- IOTA uses a quantum resistant algorithm for signature. This is a very important property but comes with the disadvantage that key reusing is insecure, therefore signing key should not be used more than once. In case the key is used another time at short time distance, it is still reasonably secure, but no more than two key uses should be allowed. We will see the consequences of this peculiarity in the next sections

How Flash Channels work The basic idea behind Flash Channels is very similar to the generic payment channel previously seen.

When a channel is created each party deposits an amount of IOTA into a multisignature address, called *deposit address*, controlled by all parties. This deposit transaction is published on the Tangle.

Once the initial deposit is confirmed the channel does not need to interact with the network until it has to be closed.

Every time a party wants to send a transaction to the other party, it creates a transaction spending all tokens from the multisig deposit address and having as output an address, called *settlement address*, belonging *only* to the receiving party. All tokens that are not sent to the output address, are sent to a multisig *remainder address*.

The sending party signs this transaction and sends it to the other party. At this point the transaction is not valid yet, because the other party's signature is still missing. When the other party receives the transaction, he can check it and, if he considers it correct, he add his signature and sends it back.

Now both parties have a valid off-Tangle transaction. Any new off-Tangle transaction is created in the same way and represents the most updated transaction of the channel. IOTA developers didn't name these off-Tangle transactions in any particular way, but, in analogy to the Lightning Network, we will call them *commitment transactions*.

The channel is bidirectional, so even the second party can send a transaction to the first one. In this case the new commitment transaction will have three outputs: the first and second party's settlement addresses and the remainder address.

When all parties decide to close the channel, they create and sign a *settlement transaction* that distributes all tokens of the deposit address to the settlement addresses, leaving zero tokens in the remainder address.

In this way all tokens go back to addresses that are not multisig, but owned by just one single party each.

This is the preferable way to close the channel because no tokens remain staked in the deposit or remainder address. Anyway it's possible that one or more of the parties don't come to an agreement and refuse to close the channel cooperatively.

That's the case of unilateral channel closure.

In any moment all parties have a set of valid commitment transactions. Whenever they want, they can decide to unilaterally close the channel by publishing one of these transaction on the Tangle.

Once one of these transactions is published on the Tangle and confirmed, all other off-Tangle transactions that the parties exchanged during the channel lifetime become invalid, because they all spend from the same deposit address.

As a consequence the channel is closed and possible tokens that are still staked in the remainder address keep on being unspendable by all parties. They can be considered lost, unless all parties subsequently come to an agreement and create a transaction assigning that tokens to one or more addresses belonging to just one person or entity each.

Incentives to behave honestly without enforcing timestamps Like for any payment channel in a trustless decentralized environment, we have to consider the possibility that some party doesn't behave honestly.

The most trivial way a malicious party can try to cheat is by closing the channel unilaterally by issuing not the most recent transaction, but the one most convenient to him.

For example, if the most recent transaction assigns him 5 iota and an older one assigns him 10 iota, he could put in the Tangle the older transaction.

Another way to cheat if is one of the parties disappears: in that case all the staked funds are lost forever because the signature of all of the parties is required to redeem the funds staked in the deposit address.

In Lightning Networks these problems are solved by using mechanisms involving time locks and revocation keys. All these mechanisms assume the enforcing of timestamp in the transactions.

Here comes into play one of the key differences between Bitcoin and IOTA that we previously underlined: at the moment IOTA does not enforce timestamps. The Tangle hasn't got a plain sequence of blocks like the blockchain, so it's not possible to guarantee with absolute certainty the chronological order of each transaction with respect to any other. IOTA developers are working to enforce a confidence interval in the timestamp but it will probably never be possible to guarantee a precise affordable timestamp like in blockchain based cryptocurrencies.

Therefore Flash Channels developers used a different approach to provide economic incentive to behave honestly.

The key difference between Flash Channels and Lightning Networks is in the commitment transactions.

In Lightning Networks a commitment transaction is a script that in any case assigns *all* staked funds to the participants of the channel. There is no multisig remainder address so no matter which commitment transaction is published in the blockchain, all fund will be distributed among the participants.

Instead in Flash Channels every commitment transaction assigns *only a part* of the staked funds to the parties. The remaining part is assigned to a multisig remainder address neither of the participant can spend from without the permission of the others.

It's important to remark that at every new commitment transaction the amount of tokens in the output addresses of the participants to the channel increases or remains unchanged, but never decreases.

When a selfish party wants to unilaterally close the channel for its own benefit, it has to decide which commitment transaction to publish.

The only parameter he looks at is the net difference of tokens owned by him between when he closes the channel and when he entered the channel.

The amount of tokens used to fund the channel is a fixed parameter chosen when the channel is open, therefore the only parameter that can change in commitment transactions is the tokens in his settlement address.

As we said above, this parameter never decreases in new commitment transactions, therefore the most recent transaction is the one with the best outcome for both parties.

This act as an incentive to publish, in case of unilateral channel closure, the most recent transaction, which represents the current state of the channel.

Reducing amount of transactable tokens as an economic incentive As a further incentive for parties to behave honestly, IOTA developers decided to add another feature to the commitment transactions, that has no equivalent in Lightning Networks.

This incentive consists in reducing the total amount of transactable tokens within the channel.

According to the developers this reduces the incentive for a party to attach a previous transaction bundle as they will never achieve a better outcome than the channel's most recent state.[20]

Let's try to illustrate the mechanism with an example.

Alice and Bob fund a Flash Channel with 50 iota each, so the total amount in the deposit address is 100 iota.

After that, Bob buys a sandwich costing 5 iota from Alice. The commitment transaction created by Bob to pay Alice is something like that:

- -200 Deposit Address
- 10 Alice output
- 0 Bob output
- 190 Remainder Address

As we can see, Alice was supposed to receive 5 tokens, but she actually received 10 tokens. In the generic case, if a party sold a good or a service worthing N tokens, he will actually receive 2N tokens.

This is the explanation provided by the creators of this mechanism: all those tokens are taken from the multisig deposit address, where each party had put an equal amount of tokens, therefore the 10 tokens received by Alice can be thought of like 5 coming from Bob's deposited funds and 5 coming from Alice's deposited funds.

By "burning" or returning the same proportion of tokens to the recipient from its own stake, the latest bundle in the channel is supposed to have the best outcome for both recipients. No previous bundle will yield more tokens than the current one will.

I personally don't agree with this design choice and the explanation they provided to justify it.

It's true that the most recent transaction will always have the best outcome for all participants, but that's not because of the choice to send 2N tokens in each commitment instead of N.

Let's see another example and consider two possible different design choices for Flash Channels.

Alice and Bob open a Flash Channel by funding it with 200 iota each, so the total amount in the deposit address is 400 iota.

CASE 1 (sending 2N tokens for each commitment, as it actually happens)
Alice buys a bottle of wine from Bob for 100 iota.

commitment #1

- -400 Deposit Address
- 0 Alice output
- 200 Bob output
- 200 Remainder Address

After that, Bob buys a protein shake from Alice for 70 iota.
commitment #2

- -400 Deposit Address
- 140 Alice output
- 200 Bob output
- 60 Remainder Address

In the second commitment Alice's output increased, while Bob's output didn't change.

In case of unilateral closure, Bob has no preference on which commitment to publish, while Alice has the best outcome in the most recent transaction.

CASE 2 (sending just N tokens for each commitment) Alice buys a bottle of wine from Bob for 100 iota.

commitment #1

- -400 Deposit Address
- 0 Alice output
- 100 Bob output
- 300 Remainder Address

After that, Bob buys a protein shake from Alice for 70 iota.

commitment #2

- -400 Deposit Address
- 70 Alice output
- 100 Bob output
- 230 Remainder Address

As before, in case of unilateral closure, Bob has no preference on which commitment to publish, while Alice has the best outcome in the most recent transaction.

This example shows how sending 2N tokens instead of N is not an incentive to publish the last commitment in the Tangle, because this happens anyway. In both cases the incentive they have to publish one commitment rather than the other one is the same.

What changed between CASE 1 and CASE 2 is the amount of tokens in the multisig remainder address. Keep in mind that in case of unilateral channel closure, tokens inside this address are not spendable by any participant, and will stay there forever unless the parties eventually find an agreement to distribute the tokens in addresses owned by single entities.

Commitments in CASE 2 leave less tokens in the remainder address and this can be seen as an advantage by both parties because it means that they will have more spendable tokens.

Another interesting observation is that in CASE 1 the amount each party deposits at the beginning is the maximum amount of tokens each party is able to spend in the channel. This sets an upper bound to the number of tokens that a party can spend, avoiding that he spends more than what he initially put in the channel.

That's the only way i can interpret this "2N method" as an incentive for the participants to behave honestly.

Moreover, using the "2N method" halves the total amount of spendable tokens in the channel with respect to using the "N method".

For example in a channel funded with 50 iota by each of the two parties, the total amount of spendable iota is 50, while using the "N method" this amount would be 100.

Finally, it does not solve completely the fundamental problem of giving an incentive for both parties to always publish the most recent transaction.

Coming back to the previous example with Alice and Bob, in CASE 1 we can notice that in both commitments Bob's output is 200 iota. Therefore Bob has no incentive to publish the most recent transaction in case of dispute because even if he publishes commitment #1, his outcome is the same.

On the contrary, Alice has a clear preference on which commitment to publish because commitment #2 assigns her 140 iota more than commitment #1.

Bob could cheat her by publishing commitment #1, even if he wouldn't have any benefit from this decision.

Tree topology In the most basic approach that one could think, every commitment transaction is a new transaction having as input the same deposit address and distributing tokens in output addresses in a different way with respect to previous commitments.

Every input of a transaction needs to be signed with the corresponding private key. As a consequence, using this approach, the private key associated to the deposit address would have to be used as many times as the number of commitments that are exchanged during the channel lifetime.

One of the major peculiarities of IOTA is the use of a quantum proof signature scheme called Winternitz.

Every time this algorithm is used to sign a transaction, it reveals a large part of the signer's private key. Therefore reusing private key is considered highly insecure, because the probability of forging increases exponentially at every reuse.

IOTA mathematicians released a paper where they make some basic estimates on the risk of reusing the Winternitz signatures.

The conclusions of this paper are that it is safe to keep tokens on an address from which there was only one spending, since it is virtually impossible to forge once-used signatures. After the second outgoing transaction (which also transfers all funds to a new address) is issued, an attacker would have to do around 2^{32} iterations to be able to issue a double spending transaction; provided that the legitimate transaction is quickly confirmed, the attacker's transaction would have very little chance to win the competition against the legitimate transaction.[21]

Flash Channels are designed to have a finite, possibly not too long lifetime. Therefore using the same key twice is supposed to be reasonably secure.

Anyway in the basic approach depicted above, the key is not used just twice, but as many times as the number of commitment that are exchanged.

For this reason, this approach is not really feasible and a new solution where keys are used no more than twice is needed.

To overcome this problem, IOTA developers decided to use a binary tree topology to represent commitment transactions.

The root of the tree is the deposit address. Every node represents a multisig address created cooperatively by the participants, like the deposit address itself.

In a generic commitment transaction, tokens "flow" from the deposit to a leaf and from that leaf to the output addresses belonging to the participants.

Every new commitment traverses the tree from the root to a new leaf, so that every path is different than all the others.

Let us assume that Alice and Bob open a channel and fund it with 100 iota each. Those 200 iota go into the deposit address, that is represented by the root of the binary tree.

This channel has been open because Alice wants to pay per second a streaming service offered by Bob. Every second Alice pays 5 iota and receives a second of TV show.

When the first payment has to be made to Bob, Alice and Bob sign not a single transaction, but a list of transactions that make tokens flow from the root to Bob's output.

This scenario is represented in the image below.

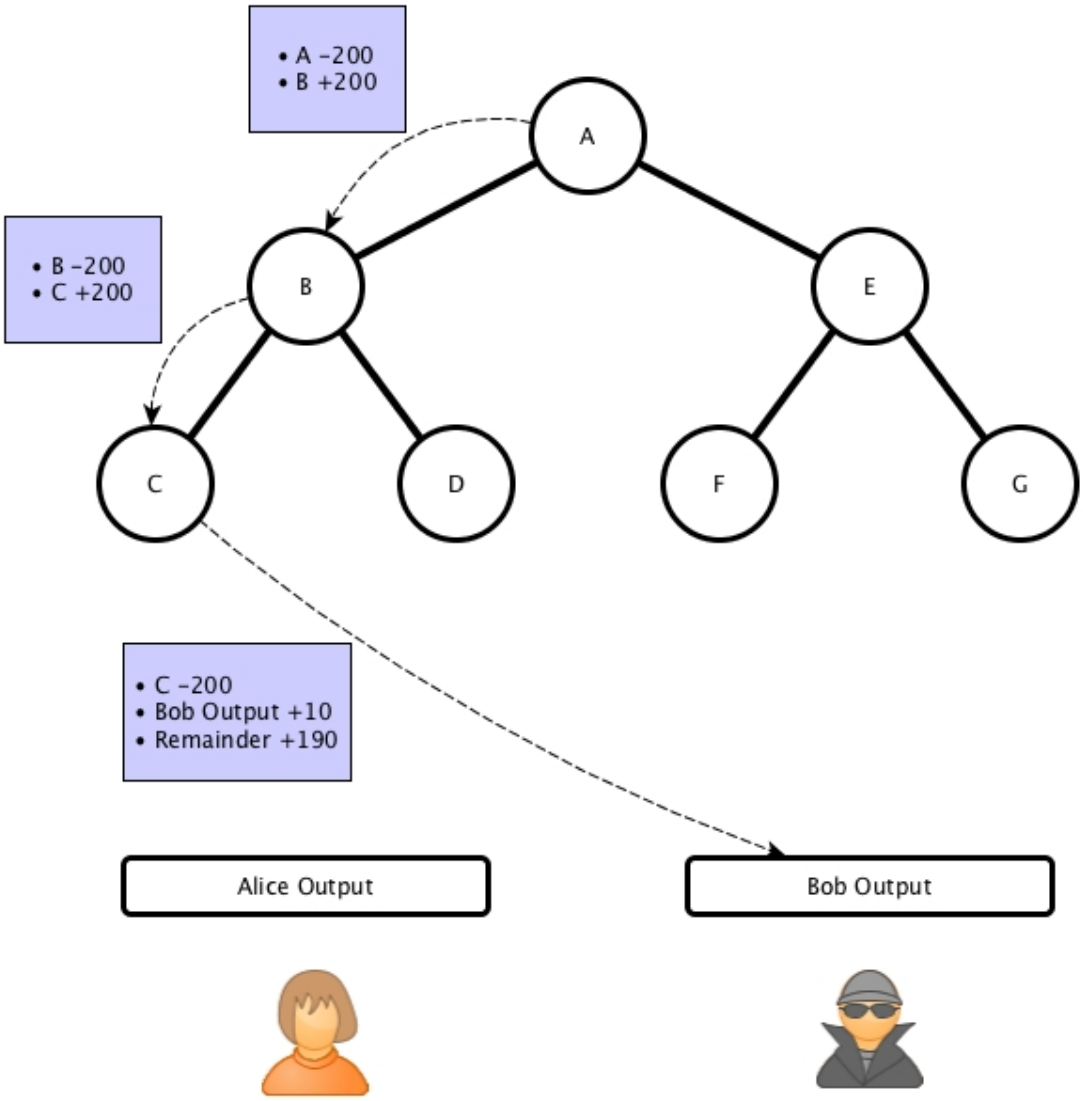

Figure 32: Example of tree topology. Nodes from A to G represent multisig addresses. Dotted arrow lines are used to indicate money "flowing" from a node (address) to another. Violet rectangles are a simplified representation of such transactions.

In the picture we can notice that 3 transactions are created:

- **tx #1:** tokens flow from root A (deposit address) to the child node B
- **tx #2:** tokens flow from B to C
- **tx #3:** tokens flow from C to two different addresses. The first one is Bob's output address, that receives 10 iota because of the 2N method. The second one is a multisig remainder address that keeps remaining tokens staked and acts as an incentive for both parties to close the channel cooperatively. If they don't, all tokens in the remainder will remain unspendable.

At this point, using the tree looks like an useless complication: we passed from having a single commitment to having a list of transaction representing the commitment!

But the real advantage comes to light when a second commitment is issued by Alice to pay Bob for another second of TV show.

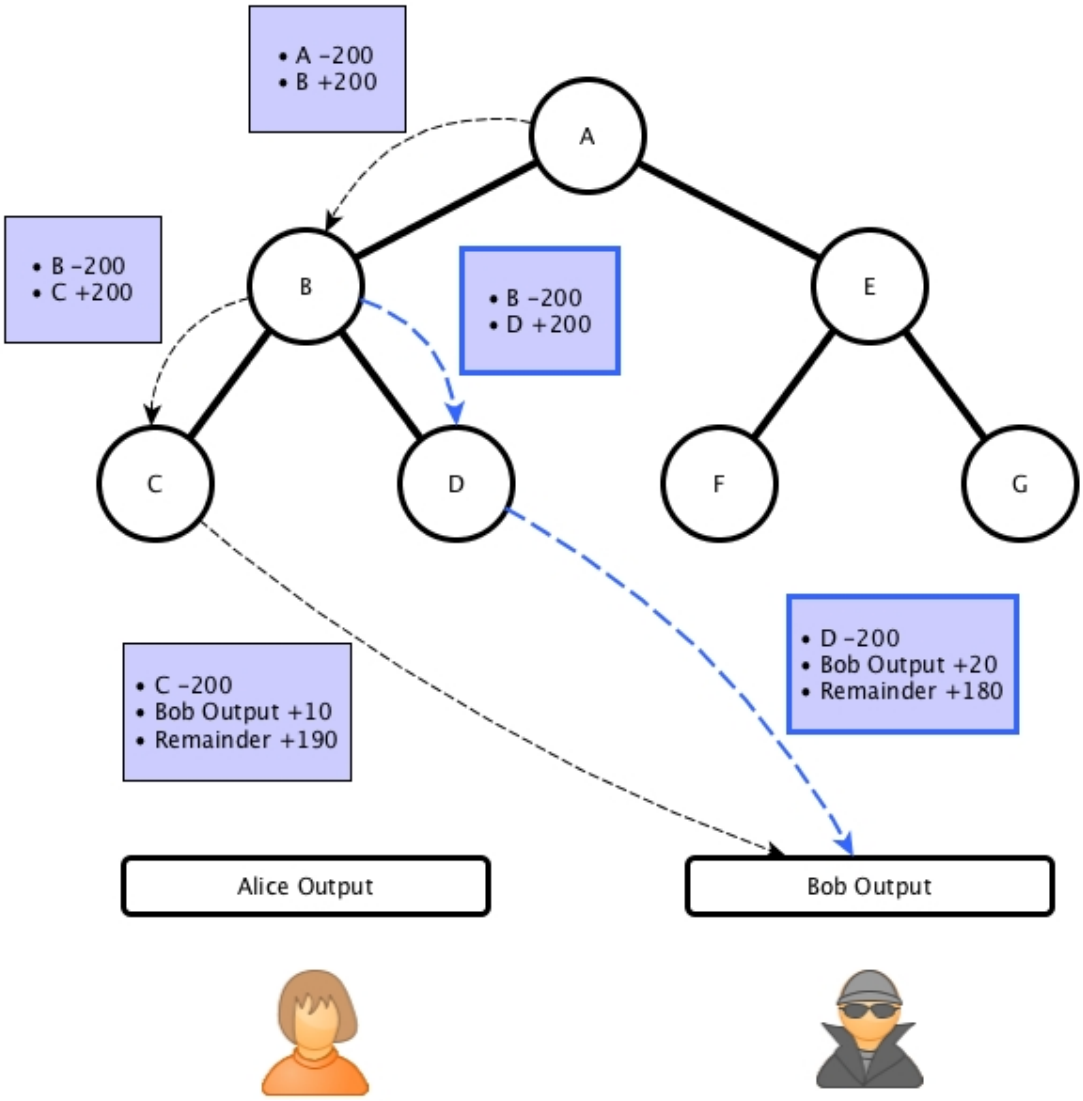

Figure 33: Binary tree after second commitment. Blue lines and rectangles with blue borders represent new transactions that were not present in the previous commitment.

This second commitment should pay 10 iota to Bob (5 for each second of video). As the previous one, it is composed by 3 transactions:

- **tx #1:** tokens flow from root A (deposit address) to the child node B
- **tx #2:** tokens flow from B to D

- **tx #3:** tokens flow from D to Bob's output and the remainder address. This time 20 iota go to Bob and 180 iota go to the remainder. In this example only Bob receives tokens, but in a generic case there could be N participants to the channel, each one receiving some tokens in their output. In this generic case the number of output addresses of the transaction would be N+1 (all the participant's outputs plus the remainder)

The fundamental transaction to notice is tx #1: it is the same used in the previous transaction! This is the reason why a tree approach solves the key reuse problem.

The private key associated to the deposit address is used just two times in all channel lifetime.

Every transaction passing through B uses the transaction from A to B, every transaction passing through E uses the transaction from A to E. This means that no matter the number of commitments that will be exchanged during the channel lifetime, every private key will be used at most two times.

This, according to the IOTA foundation mathematicians, guarantees an acceptable level of security, especially because a Flash Channel is designed to have a finite lifetime, therefore an attacker trying to take over a private key that has been used twice, has limited amount of time to do it. After the closing transaction has been published in the Tangle and confirmed, the attacker will not be able to double spend anymore.

The usage of a binary tree structure sets an upper bound on the number of commitments that can be exchanged in the channel.

A tree of $depth = N$ has 2^N leaves and that is the maximum number of commitments that will ever be possible to exchange.

Multiparty channels Flash, at its core, is a set of rules governing how parties interact with the multi-signature features of IOTA. Given this there is no limit to the number of users in a Flash Channel. This means we can have a consensus channel (all users agree) or a M of N channel (at least M of the N users agree).

A consensus channel would be good for logistically complex interactions with a large number of interrelated parties, while M of N would work well for interactions where an arbiter could be present to help resolve disputes in the channel.[22]

Staking the channel So far we always assumed channels equally staked by all participants. This is the best option when entering in a bidirectional (every party can send and receive commitments) channel with untrusted parties, because any misbehavior or disagreement would result in all parties being in an equally bad position.

Anyway there are no rules on the stake proportion: all parties are free to choose the amount to deposit in the channel, based on the trust they lay in the other parties involved.

In some particular cases equally staked channel are not the best option. For example a massive centralized server running a consumption service or an instant payment network will have to bear the burden of thousands of stakes with customers.

Such a server could not have enough tokens to fund all of the channel it has to keep open. Therefore a better option would be to have a channel funded only (100% - 0%) or mostly (70% - 30%) by the customer.

Of course the server wouldn't have an economic incentive to close the channel cooperatively because only the customer would lose tokens, but usually centralized services are based on reputation. If they cheat some customers by making them lose their staked funds without a good reason, they will lose reputation. Moreover such services are juridic entities, therefore customers would have someone tangible to sue in case of controversy.

Example of such services are EV charging stations, instant payment brokers and streaming services.

Flash Channels Javascript Library At the time of writing there is only an official Library for the Flash Channels, which is written in Javascript. Its name is *iota.flash.js* and it is available in beta version on Github.

It allows to accomplish basic operations such as opening and closing cooperatively a channel, creating commitments and updating the Flash Object of the channel.

Before analyzing in detail these operation, let's clarify what we intend as a Flash Object

Flash Object We said that a Flash Channel is essentially a shared state among all participant. In every moment all parties should have the same state stored on their device.

This shared state in *iota.flash.js* library is called Flash Object. It contains all the variables and data structures necessary to create new commitments and verify that the received commitments are correct.

We now describe the implementation of the Flash Object provided by the IOTA developers. It's important to remark that it's not the only possible one, but it's the recommended one to interface correctly with the *iota.flash.js* library.

To better understand how it works let's suppose a basic scenario where Alice and Bob open a channel by funding it with 50 iota each and after a while Alice pays Bob 5 iota.

The fields composing the Flash Object are:

- **Balance:** *Integer.* Total amount of tokens that have been staked by the participants at the opening of the channel. In our case it's 100 iota.
- **Deposit:** *Array of Integer.* Each element represent the remaining staked funds of the party. At the opening of the channel it is [50,50]. After Alice pays Bob it will be [45,45] because for the "2N method" 5 iota are subtracted by each deposit so that the channel remains equally staked. This array doesn't represent a valid transaction and its value can be easily changed by a malicious party.
- **Deposit Address:** *String.* Multisig address funded by the parties at the opening of the channel.

- **Outputs:** *JSON Object.* It's composed by (address, amount) key-value pairs where address is the output address of a party and amount is the amount of tokens currently received by the party. In our example it is initially void; after the first commitment it becomes {Alice:0, Bob:10}.
- **Remainder Address:** *JSON Object.* It represents the multisig address in which are deposited the tokens that are not sent to outputs in a commitment.
- **Root:** *JSON Object.* It's the most complex object of the data structure. It represent the binary tree in a hierarchical way. Every node can have from zero to two children, each representing a child node. Only leaves have no children. In addition each node has a multisig address and a structure representing the bundle spending tokens *from* that multisig. The root of the tree is the Deposit Address.
- **Settlement Addresses:** *Array of String.* Each element is the address owned by a single party in which are put the tokens received by him in the commitments.
- **Signers Count:** *Integer.* Number of signers (participants) in the channel.
- **Transfers:** *Array of JSON Object.* It contains all the commitments exchanged in the channel. If a malicious users tries to modify one of these commitment, it becomes invalid.

Opening a Channel Once the depth of the channel, the initial deposit amount of each user and the final settlement addresses for the participants have been entered, an initial transaction is created within the Flash channel. This transaction generates bundles for each node down the tree and then a final bundle with all of the channel's IOTA deposited into a remainder address.

At this point a deposit address (root of the binary tree) is displayed. Users must deposit the amount agreed upon and wait for the transfers to published on the Tangle and confirmed before they start transacting in the channel.

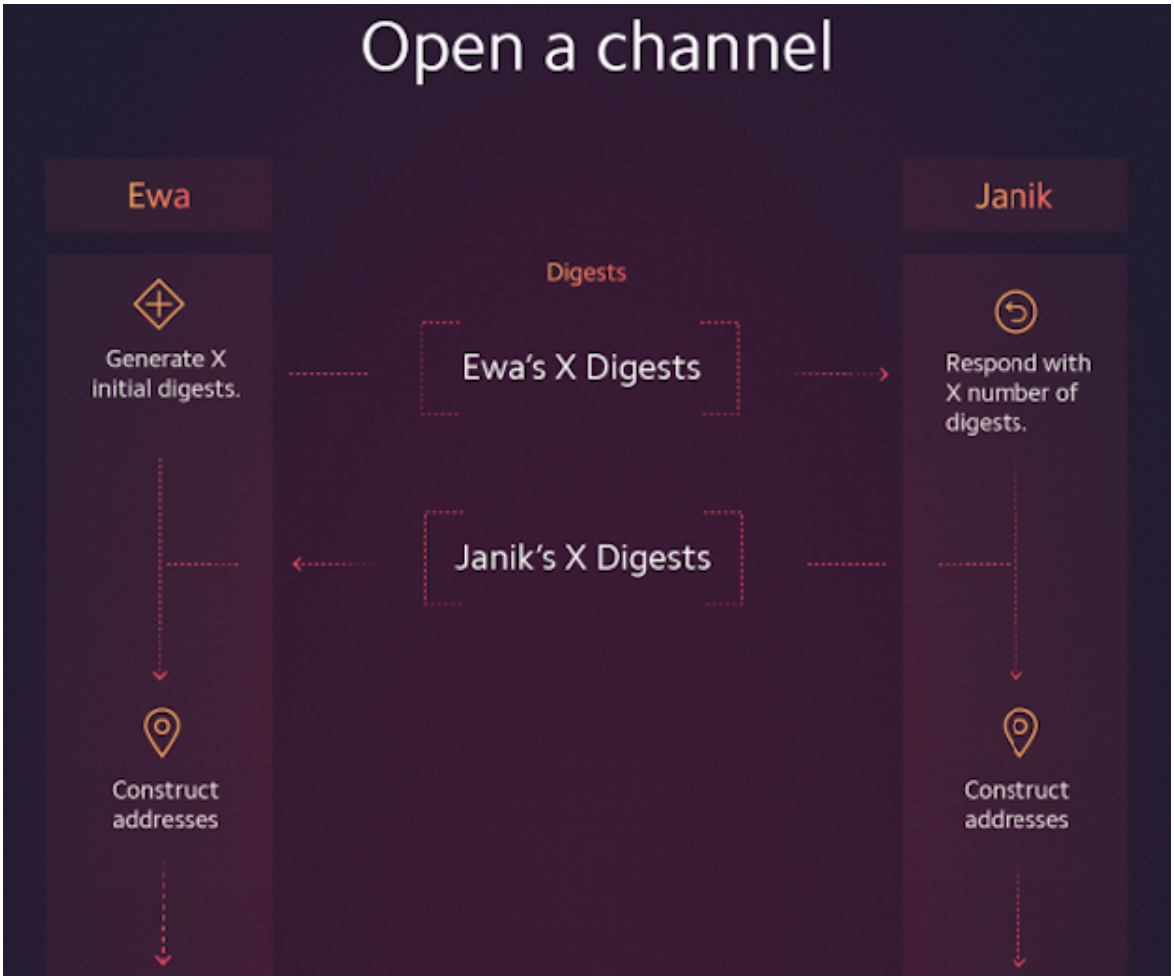

Figure 34: First step of a channel opening is the creation of a set of multisig addresses representing the nodes of the tree

Let's look in detail how this operations happen.

The first step is creating the tree (multisig addresses). To create it, it's first necessary to create the digests from which can be extracted the multisigs. Since creating them is time consuming (each digest requires at least 2 seconds on an average PC) only the nodes from root to the first leaf are created. Other nodes of the tree will be created when necessary.

The second step is creating a transaction having as inputs the participant's addresses and as output the deposit address. This transaction is signed by all parties and published in the Tangle.

At this point parties should wait for the transaction to be confirmed before starting to create commitments.

Creating a commitment When creating a new commitment in the channel a user must build a list of bundles and propose it to the other parties in the channel. After reviewing and accepting the proposed list of bundles, the users will generate and return the signatures. Once all parties have each other's signatures they validate the bundles' signatures and then change their local channel balances.

The list contains a set of bundles moving tokens from the root of the tree to a leaf and from the leaf to user's outputs and remainder address.

When constructing a new commitment the Flash library will update the Flash Object. Those are the changes that are made:

- tokens are subtracted from the Deposit array elements.
- tokens are added to the Outputs array elements representing the recipients of the current commitment.
- a bundle is added to every hierarchical element of the Root Object. This bundle is part of the list of bundles representing the commitment. This is made because every element of the Root Object represents an address, therefore it has to keep track of the bundles that spend tokens from that address.
- the commitment is added to the Transfers array.

When transacting in the channel each user has an equal right to refuse the commitment occurring. If a user receives a proposed commitment that they do not agree with, they reject it by not signing the list of bundles.

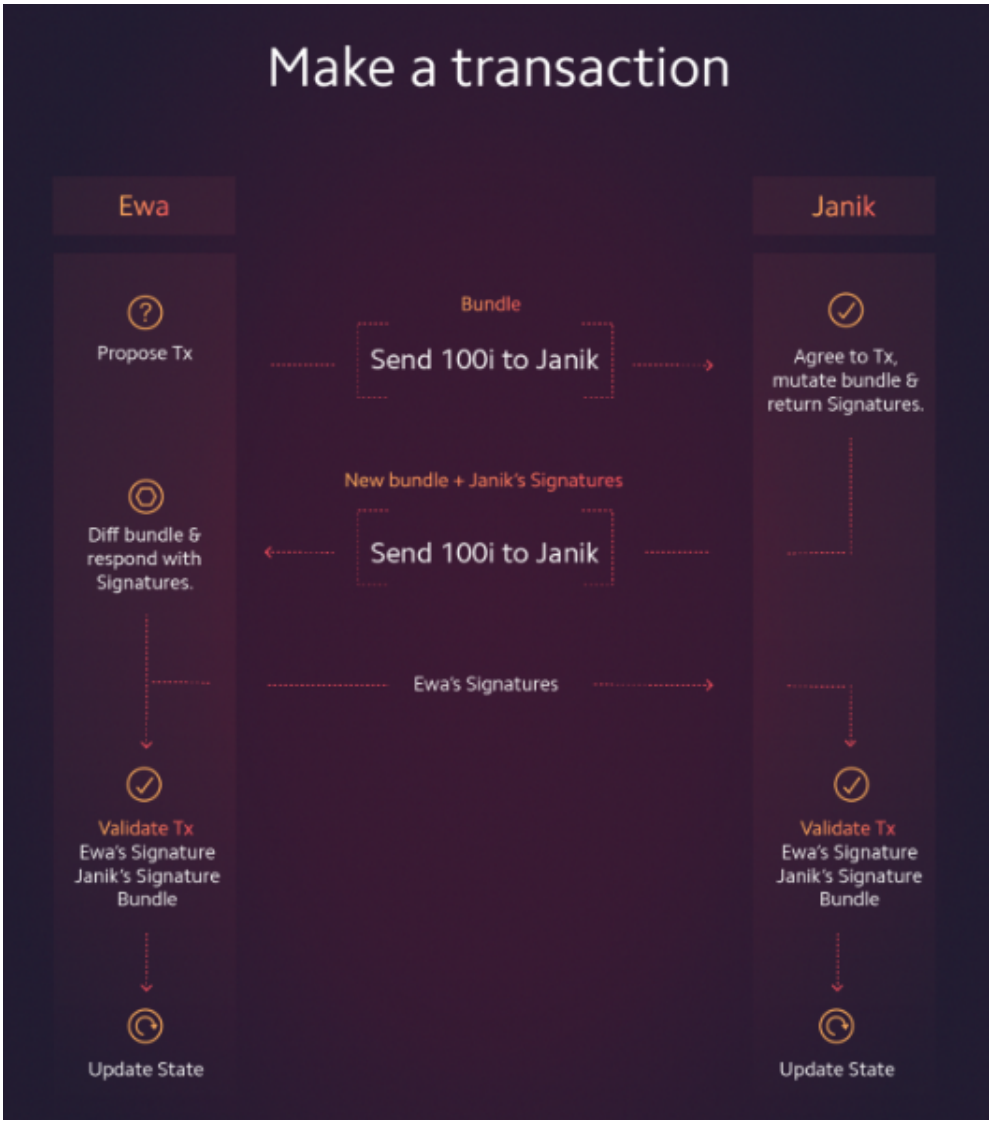

Figure 35: Step by step construction of a new commitment

Closing a channel cooperatively To close a channel, the proposing party constructs a list of bundles that does not propose a new value transfer. Instead it takes the remaining channel balance and divides it among the channel's parties according to the proportion of the Deposit array when the channel was opened. During this process the party finds the minimum amount of bundles required to close the channel.

The other parties in the channel check the list of bundles and return their signatures or reject it. This list of bundles is then published on the Tangle.

At the moment of writing the *iota.flash.js* library has no API to close the channel unilaterally. Anyway it could be implemented in a quite easy way by simply taking the desired commitment from the Transfers array and publishing it on the Tangle using *iota.lib.js*, the official Javascript library for the IOTA Core.

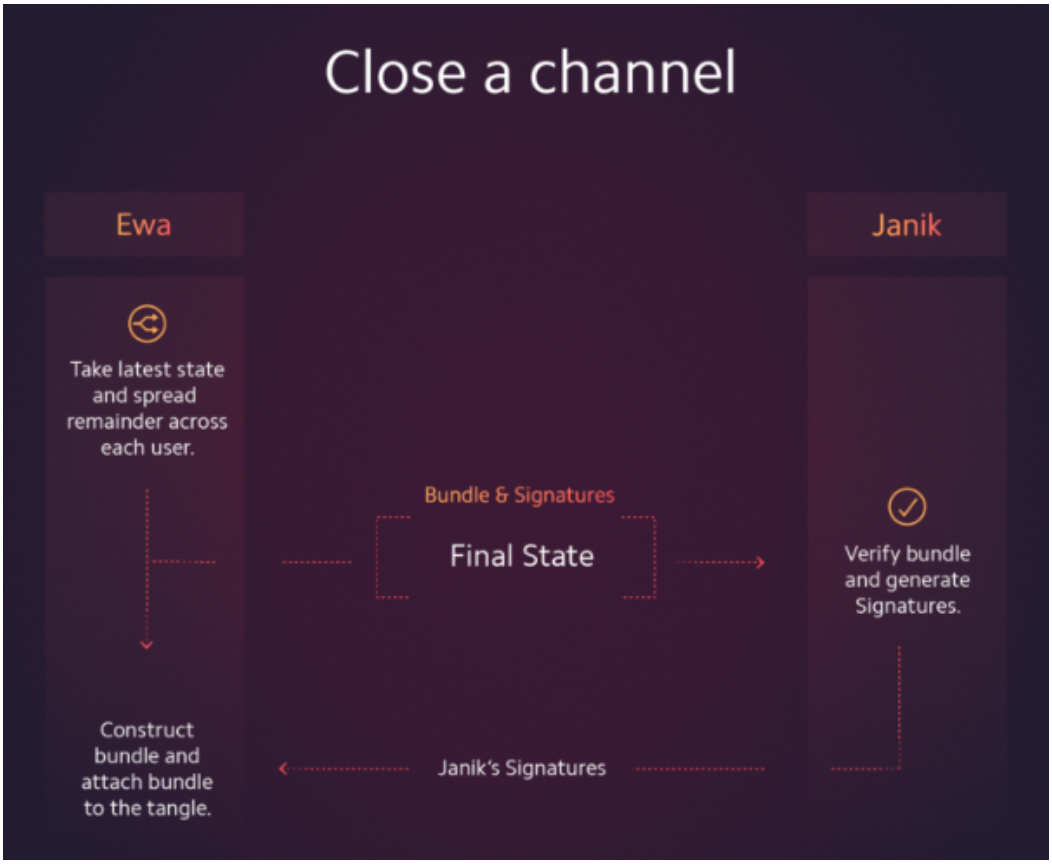

Figure 36: Step by step construction of a new commitment

Applications Flash Channels enable very fast, theoretically sub-second, bidirectional payments among two or more parties. This, together with the native IOTA feature of the lack of fees, allows a large set of applications.

It fits particularly well in all use cases dealing with data or resource streaming because they require instant, fine granularity and high throughput payments.

Some of these are EV (Electric Vehicle) Charging , Bandwidth on Demand, Resources on Demand (such as Computation, Storage etc.), Pay per Article and many others.

Recently an Hackathon on Flash Channels has been announced by the IOTA Foundation. This gave the possibility to community member to propose and realize new ideas concerning Flash Channels.

The most interesting ones were a Fognet prototype using payment channels in resource and bandwidth restricted environments to exchange data for instant payments; a generic Home Assistant allowing a user to monetize IoT devices e.g. a coffee machine attached to the automation system; a real-time data marketplace using pay-per-stream policy; a WiFi Access Point that uses Flash Channels payments to charge users for their access to data by the minute; an application implementing Flash Channels between Google Cloud and a cross-platform mobile application to enable users to sell their data to interested parties.

As we can see a lot of really different applications can benefit from Flash Channels. Another field where, according to me, they have great potential is Smart

Mobility. In next section we will discuss the concept and implementation of an application using Flash Channels in this field.

3.4 Comparison between Lightning Networks and Flash Channels

Now that we explained how Lightning Networks and Flash Channels work, we can make some consideration about the similarities and differences between them.

The biggest difference, that has many consequences on the implementation choices of the channels is the fact that Bitcoin enforces timestamps while IOTA doesn't.

This obliged IOTA developers to look for novel solutions to try solving the problem of guaranteeing that untrusted parties behave honestly.

To better organize this section, we first discuss the similarities between Flash Channels and Lightning Networks, then we discuss the differences, explaining advantages and disadvantages of the two technologies.

Similarities:

- Both of them are implementations of the same concept of Payment Channel that allows to exchange transactions outside the DLT in a secure way by publishing only the opening and closing transaction.
- Both of them enable instant and high throughput transactions
- Both of them manage, even if in different ways, both the scenario of cooperative and unilateral channel closure
- Both of them consist in a shared state stored locally on each participant's device and updated at every new commitment transaction
- Both of them make extensive use of multi-signature
- Both of them are Layer-2 protocols built on top of a decentralized payment infrastructure

Advantages of Lightning Networks over Flash Channels:

- Incentives provided by time locks and revocation keys are much more effective in preventing misbehavior by the parties than the "2N method" proposed by Flash Channels. Indeed this second mechanism doesn't avoid that one party cheats the other by publishing a prior commitment whose outcome is the same for the cheating party and worse for the cheated party.
  Let us assume a payment channel is funded by Alice and Bob with 50 tokens each. If Alice pays Bob 5 tokens, in the Lightning Network this will result in a commitment assigning 55 tokens to Bob and 45 to Alice. If one of the parties disappears or tries to cheat, it will be punished so that the cheated party will never be damaged. The same scenario in Flash Channel would result in a

commitment assigning 10 tokens to Bob, 0 to Alice and 90 to the remainder. If Alice disappears, Bob will be obliged to publish the commitment, but instead of having a net gain of 10 tokens, he will have a net loss of 40 tokens (he had put 50 tokens in the channel when he had started). Of course Alice has a worse outcome that should prevent her from misbehaving, but there is a scenario where the cheated party is damaged, while in Lightning Networks there is not.
Another way to look at the problem is that in Lighting Networks commitment transactions leave no tokens in the multisig input, therefore unilateral channel closure does not result in the loss of money for both parties. In Flash Channels none of the commitments will ever result in a net gain of tokens for any of the parties. As a result unilateral closing is never an option in Flash Channels, while in Lightning Network it is, even if it's preferable to close it cooperatively.

- In Lightning Network the channel can remain open indefinitely and can have billions of commitment transactions. In Flash Channels the maximum number of commitments depends on the depth of the tree that is set at the opening and cannot be subsequently changed. In addition to this, opening a channel with a high depth slows down the opening of the channel and the creation of commitments, because more multisig addresses (nodes) have to be created and this, in the current Javascript implementation, is a bottleneck.

Disadvantages of Lightning Networks over Flash Channels:

- Lightning Networks is based on Bitcoin blockchain therefore it has fees. They have to be paid to the miners and to the intermediate parties that connected the the two or more end users of the channel. As a consequence opening and closing a channel has a cost. Therefore Lightning Networks are not usable in scenarios requiring to open and close channels frequently. On the contrary Flash Channels is based on IOTA Tangle that is feeless, so opening and closing channels has no cost. This partially cancels the disadvantage of having a finite number of exchangeable commitments in the channel. Anyway opening and closing a channel requires time to compute the PoW on the bundle to be put in the Tangle and expecting for it to be confirmed.

- Lightning Networks commitments subtract tokens to the miners, so they could be less incentivized to keep mining. IOTA has no miners so using Flash Channels doesn't have this kind of bad consequences of the network.

- Lighting Networks are based on routing. According to a skeptical view, routing is likely to be a significant problem, since finding a short path between parties is a difficult problem to solve algorithmically. If no route is found, the parties will have to engage in the cumbersome process of selecting an on-chain transaction by manually changing the payment process.[23]
Moreover the network could centralize around a few large hubs as this could be the most efficient model.
Centralization is the exact opposite of the reason why cryptocurrencies are

born.
Flash Channels have no routing because they directly connect the end-points.

- Lightning Network is an ad-hoc solution to an inherently not scalable protocol. Instant transactions are not the reason why it has been created. Instead Flash Channels have been created just for that, because scalability is already guaranteed by the IOTA protocol.

4 Parking Meter Application

IOTA was born with a primary focus on enabling Internet of Things (IoT) applications. The machine-to-machine (M2M) economy, or the idea that machines would be acting directly with others without human intervention, is growing at a fast pace.

According to the IOTA founders, in the next years new business models based on decentralization will emerge in a distributed peer-2-peer (P2P) economy.

In this context, one of the most interesting topics is the future of cars and mobility in general.

The current trends goes towards Connected Vehicles, Autonomous Vehicles, Electric Mobility, Shared Mobility and Urban Mobility.

It's quite clear that future cars will actually become digital platforms with ubiquitous connectivity.

Connected vehicles are also going to support wireless transactions through digital wallets.

IOTA is very active on trying to become the economic infrastructure of this revolution. As an example ElaadNL, a dutch research center, realized a proof of concept Charge Station for Electric Vehicles (EV) running fully on IOTA for the payment of energy.

The idea is that cars will have their own wallet and will be able to perform transactions independently by interfacing with other machines, enabling the IOTA vision of Machine-to-Machine economy.

The scenarios regarding Autonomous Vehicles and Smart Mobility are many, and I decided to focus on a particularly cumbersome one: car parking.

The problem Traditional car parking has different disadvantages. The main problems are due to two factors: paper tickets and fees.

- *Paper tickets:* they are not quick to get and have no flexibility. Generally the car owner has to look for a near parking meter (if there is one), print the ticket and go back to the car to expose it. It could seem a small amount of time, but in a fast paced world, even saving some minute can be important. The most crucial problem is however the fact that the customer has to decide in advance how much he's going to stay in the parking lot. If he selects a too small amount of time, he risks to be uncovered after the expiry, therefore he usually decides to pay for more time than he is actually going to stay. If he prefers to pay less, he risks to have to go back to the parking lot to extend the expiry date when it's near to the end. This results in a useless loss of money and time.

- *Fees:* parking meters usually have inconvenient fee policies. One of the most common ones is to pay every 30 minutes: this time interval is too large to allow people to pay for the real time the stay in the lot. Most of them will pay more or will decide not to pay at all for a brief period of time, even if they risk a penalty.

4.1 The concept

The idea of my application is to use Flash Channels technology to implement a pay-per-use policy for car parking.

The participants to the channel are the Car (acting as a client) and the Parking Meter (acting as a server). Both of them have their own wallet, so the payment is Machine-to-Machine.

The server is an application running on the Parking Meter hardware, while the client is an application running on the car computer.

When the car arrives in the parking lot, it's possible to select the lot on the application screen and open a channel with the Parking Meter wallet.

The customer just decides the maximum amount of time he wants to leave the car parked (and how many iota to put in the channel as a consequence).

When the user presses the button to start parking, commitment transactions start to be sent on a regular interval of 1 minute. Every time the server receives a correct payment, it issues a ticket valid for that specific minute, signs it and sends it to the client. In this way the client has the proof that the service has been provided and can keep on paying until it desires.

After having started parking, the customer leaves the parking lot and in any moment he can check the parking status (is parking, how much has been spent so far and other informations about the parking lot) on the application.

When the customer goes back to the parking lot, he just closes the channel by pressing a button and leaves the lot.

The emission of tickets also allows easy check by the law enforcement officer. For example a police officer can check in any moment whether or not a parked car is actually paying by simply querying a database containing all the tickets that have been issued. To identify the car it can use the plate or, to speed up the process, a QR code representing the plate. This part has not been implemented, but it has been reported here to complete the theoretical concept of the project.

4.2 Advantages over traditional parking meters

This novel approach allows to solve the traditional car parking problems. There are three main advantages:

- *Customers can pay for the actual number of minutes they stay in the lot.* No more 30 minutes interval will allow customers to pay less and parking lot administrators to decrease the rate of non-paying customers. Indeed they will have an incentive to pay because of the new more convenient fee policy.

- *The system can be applied not only to controlled parking lots but also to parking lots without gates.* Every police officer with a mobile phone can check if the customer is paying or not. Some of the existing electronic car parking services such as Telepass only work in controlled parking lots.

- *Customers don't need to extend the parking expiry date.* It never expires until they come back to the car (unless they spend all the money they put in the channel). They can safely put in the channel an amount greater than what

they are going to spend because they know that if they come back to the car before the expiry time, all the amount of money that has not been spent will be paid back to them at the closure of the channel.

- *No user registration.* Electronic car parking applications usually ask the customer to register and send them personal informations. Depending on the kind of contract between user and service provider, this data could be sold to third parties without the customer being completely aware of the details of this process (what data are sold, to who, for what purpose, etc...). This application aims to be as anonymous as possible by using the native IOTA feature of pseudonymity (the end points are identified by their addresses, from which it's not possible to trace back to the real person). Of course total anonymity is not achievable because every car is identified by its license plate that is directly linked to its owner. Possible alternative solutions are proposed in the *Further Developments* section.

- *No need to trust the parking lot administrator.* Most of parking lots are operated by the Municipality, a company or legal entity. They rely on a reputation and in case of controversy they can be sued. This means that cheating the clients is not in their interest and lack of trust should not be an issue. Anyway in some particular situations the parking lot owner could be unknown or a single untrusted person. Also in these cases the system can be applied because Flash Channels are supposed to provide incentives for all parties to behave honestly. If the server doesn't send the client the ticket it expects, the client can immediately stop the payments.

4.3 Architecture

We explained in the previous sections how Flash Channels enable instant and high throughput transactions between two parties.

This perfectly fits with the requirement of the car to pay with a fine granularity.

My initial idea was to set a pay-per-second policy, but using the *iota.flash.js* Javascript library i found out that every off-Tangle transaction required at least some second to be performed. Therefore i switched to a pay-per-minute policy. I think this granularity is fine enough to allow customers to pay a fair fee.

Another aspect to take into consideration is that Flash Channels allow to exchange just a limited number of commitments. This number depends on the depth of the binary tree it uses.

In case the end of the tree is reached, it's possible to continue the payments by closing the current channel and opening a new one. However this should be avoided because it requires time to publish the closing transaction and creating the nodes of the new tree.

In this case payments should be made every one minute and switching from one channel to another could require more time.

To avoid this situation it's fundamental to properly set the depth of the tree.

The number of commitments to be exchanged depends on the number of minutes the customer is going to stay in the lot. If the customer is going to stay N minutes,

the depth of the tree is given by the formula below:

$$depth = \lceil \log_2 N \rceil \tag{3}$$

For example if the customer is going to stay 440 minutes, at most 440 commitments will be done.

The formula becomes:

$$depth = \lceil \log_2 440 \rceil = \lceil 8.78 \rceil = 9 \tag{4}$$

With this depth, the tree will have $2^9 = 512$ leaves. This number is sufficient to make all the necessary commitments, since theoretically every new commitment uses a new leaf.

Actually, with the current implementation of the Flash Channels library, every leaf is used three times. This is probably made for performance reasons, since every new tree node creation requires a considerable amount of time (2-3 seconds on an average PC).

This is not secure, but at the moment Flash Channels are just used in prototypes running on a testnet Tangle where tokens have no real value. Therefore security is not an issue.

Furthermore, the parameter setting the number of times a leaf is reused can be easily changed by code.

The application is composed by two components:

Client It represents the car wallet. I used React.js for building the user interface. It's a nice Javascript library maintained by Facebook that allows to easily build one page web applications. It aims primarily to provide speed, simplicity and scalability. For simplicity the wallet seed is pre generated and already has some tokens in it, so there is no need to fund the wallet or manually set it.
When the user opens the application, he just has to choose the maximum amount of time he guesses to stay in the parking lot. Based on this input, the application logic calculates the amount of tokens necessary to open the channel. Customers are not used to think in terms of iota, so all the amounts are expressed both in iota and in euro.
Unlike traditional parking lots, customers can safely put a larger amount of tokens than they actually are going to spend, because they know unspent tokens will be returned to them when they will leave the lot.
It's important to remark that only the client can decide when to start and when to stop the payments. The server only handles and responds to client requests.
Every minute the client pays for the permanence in the lot and receives in response a ticket valid for that specific minute. This implementation choice is not the most trivial one: the most basic way would be to issue just one ticket when the channel is closed. Anyway my approach has two main advantages:

- it works even if the server crashes before the the channel is closed because the customer has the proof (tickets) of all the payments made until the crash. In the alternative trivial approach the customer would never receive any ticket and would have no proof of payment in case of controversy.

- it's coherent with the Flash Channel concept of pay-per-use and can also be applied to untrusted parking lots, because if the requested service (in this case the ticket) was not provided, the customer would be able to immediately stop the payments, avoiding to lose further tokens.

All ticket are cryptographically signed with the private key of the parking lot. The associated public key is sent to the client at the channel opening, therefore both the client and the police officer can check the authenticity of the ticket.

Server It represents the parking lot and is able to handle requests from multiple clients.
All the libraries and code base of Flash Channels are written in Javascript. To avoid rewriting code i chose to develop the server in Javascript using Node.js.
Node.js works on a non-blocking I/O model that makes it clean and usable, ideal for data intensive real-time applications. It has several benefits: an active community that makes easy to find informations, simultaneous requests handling, presence of many libraries to interface with most of the databases.
The server receives requests to open the channel, close the channel and make a payment. Every payment is actually a commitment transaction and as such it requires to be signed by both parties to be considered valid. The server receives the proposed commitment together with the client signatures. It checks the commitment and, if it is considered correct, add its signature to the commitment and sends it back to the client together with the ticket. In this way both parties have the same signed commitment transactions.
Every time a new ticket is sent to the client, it is also stored in a local MongoDB database on the server.

This allows both the client and the server to have their independent copy of all the issued tickets. The signature guarantees the identity of the server in case of controversy.

Communication Flash Channels don't enforce any communication protocol, so i had free choice.

I immediately excluded Bluetooth because the distance between car and server could exceed its physical range.

I chose TCP because it works on a lower layer than HTTP and allows to have higher performance.

To perform the basic operations, client and server exchanges simple JSON messages containing a command and the data. All messages contain a *channelID* that uniquely identifies the channel.

Basic Operations Those are the operations that the channel has to perform during its life cycle:

- **Open Channel:** when the client wants to open a channel, it creates all the necessary digests to go from the root to the first leaf of the tree. Then it sends a *startSetup* message to the server. This message contains the just created

digests, the amount of tokens to fund the channel and the license plate of the car.
The server creates its own digests. Having received the client's digests, it's able to create the multisigs. The flash state containing output addresses, multisigs and other metadata is created and stored on the local server database. After that, it responds to the client with a *returnSetup* message containing its digests, its output address and the public key that the client will use to validate the ticket signature.
Now the client has all the digests pairs needed to create the multisig addresses and the remainder address, so it creates them and, similarly to the server, locally stores its flash state, that will be identical to the one of the server.

- **Send Commitment:** when a new commitment has to be created, the client checks if there are available leaves to use. If not, it creates a new one together with the server, which receives a *getBranch* request message and responds with a *returnBranch* containing the requested digests.
  After the creation of the new multisigs, a new commitment is created, signed by the client and sent to the server via a *composeTransfer* message. The server checks the commitment and, if it's considered correct, signs it sends the signature back to the client via *returnSignatures*.
  The client checks server's signature, adds its own signature and obtains a valid transaction signed by both parties. This transaction is stored on the local client database. After that the client sends its signature back to the server via *returnSignatures*.
  Now all parties have a valid commitment. The server creates a ticket valid only for one specific minute, signs it with its private RSA key and stores it in the local database. Later, it sends the ticket to the client via *returnTicket*.
  Now both parties have a valid commitment and a signed ticket.

- **Close Channel:** to close the channel the client sends a *closeChannelRequest*. The server checks if all payments from the opening to the closure of the channel have been done correctly. If some payment is missing it responds with a *payDuesRequest*, specifying how many payments are missing. The client pays by issuing a proper number of commitments and then tries to close the channel again.
  If the server agrees that all payments have been made, it responds with *closeAccepted*.
  The client provides to create a settlement transaction to close the channel, taking all tokens from the deposit and distributing them properly to the two parties. Then it signs the settlement and sends it to the server via a *closeChannel* message. If the server agrees, it adds its signature to the bundle and sends it back via *returnSignature*.
  Now both parties have a valid settlement transaction. To make the client more lightweight, i decided that the server has to publish the transaction to the Tangle. It such a way the client must not perform any Proof of Work. This is particularly useful if the client runs on a mobile or IoT device, because it saves energy and time to the customer.

So the client request the server to publish the transaction via *readyForAttaching*. When the transaction is published, the server signals the client with a *channelClosed* message.

Connection Loss Handling One of the most interesting features is that the application works even in the scenario where the TCP connection is temporarily lost.

When the user or the server temporarily goes offline, payments can't be made because TCP messages are not send or received.

To overcome this problem the idea is that in those cases the client accumulates *dues*. The server is not receiving the payments but knows that eventually they will be made, otherwise the server will refuse to close the channel and the client will lose all the tokens it deposited in the channel.

So in case of connection loss, the server keeps the channel open and keeps track of the unpaid minutes. At the channel closure, before signing the final settlement transaction, it ask to the client to pay all its "dues". Only after those payments, it accepts to sign the settlement and close the channel cooperatively.

4.4 Results and Performance Benchmark

Flash Channels have been created to allow instant transaction exchange among parties.

Theoretically the only limit to throughput should be bandwidth. This should enable sub-second streams of transaction. In my application i performed some benchmark to see what is the real throughput.

Benchmark To be nearer to a realistic worst case scenario, i assume that the customer desires to stay in the parking lot for 6 hours, corresponding to 360 minutes.

In this case at least 360 leaves are needed. The minimum tree depth to satisfy this requirement is $2^9 = 512$. This means that at the opening of the channel 10 multisigs address have to be created: 9 represent the nodes of the tree from the root to the first leaf and 1 represents the multisig remainder address.

To create a multisig, both client and server have to create their own digests. Digest creation is the real bottleneck of the current implementation of Flash Channels. This is probably due to the fact that Javascript is an interpreted language and has low performances, therefore it is not the best option for time consuming operations such as digest creation. The average time for the creation of one digest is about 2-3 seconds, but heavily depends on the hardware on which the code is running.

The following benchmark has been calculated on an average PC (dual-core 2,7 GHz Intel Core i7).

Operation	Min Time (sec)	Max Time (sec)	Bottleneck
Open Channel	20	24	Digest creation
Send Commitment	3	30	Digest creation, signature validation
Close Channel	7	34	Digest creation, signature validation, PoW
Digest Creation	0.5	2	

The most time consuming operation is Channel Opening. In case 10 digest have to be created, this will require about 10 second for each party.

Time required to send a commitment has a high variance. Again, it depends on the number of new digests that have to be created.

In the best case scenario no digests have to be created because the previous one is reused. The only time is that required for creating the commitment and sending it to all the parties. With the current implementation of the library, every leaf is used three times before passing to the next one. As a consequence, the best case scenario happens 2 out of 3 times and is the most likely one.

In the worst case scenario (it happens just once in the whole channel lifetime) a set of new nodes going from the child of the root to the new leaf have to be created. In case of depth=9, 9 new digests have to be created, which will require at most 18 seconds. After that, another time consuming operation happens: signature validation. It consists in checking the correctness of all the transactions composing the commitment before updating the local Flash Object.

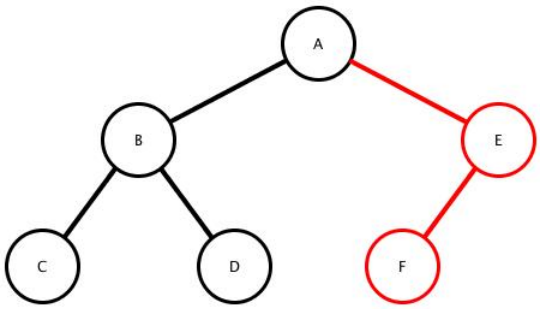

Figure 37: Worst case scenario in a simplified tree of depth $d = 2$. In this case d new digests, colored in red, have to be created. Each new digest represents a node of the tree going from the child of the root (E) to the leaf (F)

The time to close the channel is calculated in the case in which the customer has no missing commitments to pay.

In this case closing the channel simply means creating a settlement transaction and publishing it on the Tangle. The time to create the settlement transaction is the same required to create a commitment, but now the another time consuming operation happens: Proof of Work computation. In IOTA it's possible to manually set the minimum complexity of PoW, called weight. In the *testnet* the minimum

accepted weight is 9.[25] This allows, for pure development purpose, to calculate PoW in few seconds (about 3-4). This is the amount of time considered in the benchmark.

It's important to remark that in the real network, called *mainnet*, the minimum weight is 14, therefore it requires much more time to be computed (about 1-2 minutes on an average PC).

Problems and possible solutions

- *Commitments are not instant:* from these benchmarks we can see that the instant transaction objective is far from being realized in a real use case. All use cases requiring very high throughput, such as pay-per-second policies are not feasible at the moment.
  The main reason is that the current implementation of the Flash Channels is in Javascript and in beta release. In the future we could expect the release of a new library or the improvement of the existing one.

- *The parking lot has to hold a large amount of tokens:* this issue regards the way i implemented the application. The channel is equally funded. This is an advantage because it enables payments between untrusted parties but also a limitation because the server has to put in the channel the same amount as each customer. In big parking lot this could become a problem.
  Let's suppose to have a parking lot with a capacity of 3,000 vehicles and a 0.02€ per minute fee.
  In the extreme scenario where the lot is full and all vehicles have opened a channel to stay in the lot for 10 hours (600 minutes), this mean that the server must own at least $3,000 * 600 * 0.02 = 36,000$ € to keep all the channels open.
  To solve this issue it's possible to set a channel funded only by the customer. This solves the problem but requires the parking lot to be a trusted entity. It's clear that a trade off between trust and large tokens holding has to be made.

  Other disadvantages derive from intrinsic limitations of the current implementation of Flash Channels.

- *Parking duration is inversely proportional to channel speed:* the maximum number of commitments is predefined. The more time the customer desires to stay in the lot, the more multisigs have to be created and the slower the average commitment creation will become. A possible tradeoff could be opening a smaller channel and when all leaves have been used close it and open a new one. At the moment this is not convenient because of the large amount of time required to open and close channels.

- *Weak economic incentives to behave correctly:* incentives provided in Flash Channels are not as effective as in Lightning Network. In the previous sections we described how the lack of time locks and revocation keys limits the possibility to implement effective mechanism to prevent misbehavior by one

or more of the participants. In the case of my application, the customer or the parking lot could refuse to cooperatively close the channel, causing tokens loss to both of them. This could be solved when IOTA will enforce timestamps on the Tangle. According to some of the IOTA developers this feature seems to be on the roadmap but so far it's not been officially communicated whether and when it will be done.

4.5 Further developments

Starting from this Proof of Concept, a more ambitious Machine-to-Machine application could be realized.

In this application the client runs on the car's hardware and is able to autonomously take smart decisions based on data coming from the sensors inside the car.

For example it could autonomously understand in which parking lot it is located based on the GPS sensor and automatically open a Flash Channel to pay for parking when the car stops and the driver leaves the car.

When the driver goes back to the car and leaves the parking lot, the car could automatically close the channel and stop the payment. All without user interaction. Obviously the user should be able to remotely check the parking status in any moment using a mobile application that allows to see all data related to the ongoing payments and even stop the payments if necessary. In this way the user would be in total control but at the same time would have nothing to do to start and stop parking. All could happens automatically. I think this is the idea of IOTA founders when they talk about the emerging M2M economy.

Another interesting idea is to integrate the Car Parking application with another promising IOTA project: Data Marketplace. In consists in a Tangle-based decentralized marketplace where every sensor can store data and be paid for them.

The goal is to enable a truly decentralized data marketplace to open up the data silos that currently keep data limited to the control of a few entities.

With the distributed IOTA Data Marketplace, we have a shared infrastructure at our disposal to unlock data and create opportunities for it to be shared and monetized. The potential of IOTA is to help citizens in a community to retain control over their data, but to unleash its potential by making it accessible without giving up on privacy or monetization options.

By applying these concepts to the Car Parking application we could imagine a scenario where every time a car stays in a parking lot, it stores data coming from its sensors, such as position, temperature and so on, on the public Data Marketplace.

Instead of using the car license plate to identify the car (and indirectly also the car owner) we could directly identify the sensor producing data by using another module that is being developed by IOTA: Identity of Things (IDoT).[24] The idea is that every sensor should not only have its unique identifier, but also accompanying it attributes such as who manufactured it, when it was deployed, what is the expected life time cycle, what kind of sensor data is it gathering and at what granularity and so on.

In this way, every entity interested to those data could buy them directly paying

the car owner instead of the parking lot administrator. This perfectly fits with the vision of IOTA to give personal data back to the people.

Today many services monetize by selling personal data belonging to their users, that usually are not completely aware of this process. Using IOTA users could go back to being it total control of their data, deciding if and when to sell them and receiving money in return.

Data collected by car sensors could crowd-source highly accurate on-street parking informations to reduce parking-related traffic. For instance a smart city powered by the IOTA network could provide incentives for ride sharing when the parking situation is bad due to an event.[26]

5 Conclusions

IOTA is a relatively young cryptocurrency, therefore time is needed to have a complete overview of its security and to see if its expectations will actually be realized.

Anyway it undoubtedly has the potential to overcome many of the Bitcoin problems with its innovations such as infinite scalability, lack of fees, quantum resistance and partition tolerance.

These features could make IOTA become the backbone of the Internet of Things and enable a new ecosystem based on M2M economy.

Payment channels could have a major impact on applications requiring instant high throughput transactions, enabling interesting use cases related to pay-per-use, guaranteeing speed and security to all parties involved.

The IOTA implementation of this concept, called Flash Channels, is quite different compared to the Bitcoin one, called Lightning Network. This is mainly due to some differences in the two protocols and to the different structure of blockchain and Tangle.

Some potential disadvantages of IOTA, such as the usage of a one-time signature algorithm, are managed in a really innovative way using a binary tree to represent the off-Tangle transactions.

Anyway Flash Channels have some important limitation compared to Lighting Networks, mainly due to the lack of time locks and smart contracts in IOTA. This, unlike Lightning Networks, doesn't allow Flash Channels to provide completely effective economic incentives for all parties to behave honesty during the channel lifetime.

Indeed the solution proposed by the Flash Channels to overcome these issues doesn't completely solve the incentives problem, but in the future, when confidence intervals for timestamps will be enforced on the Tangle, a more secure and robust version of Flash Channels could be deployed.

Flash Channels could have a great impact on applications where a finer granularity of payments enable an improvement of the service.

An example is Car Parking. Today most of the parking fees policies are based on coarse granularity intervals (e.g. 30 minutes). Introducing pay-per-use policies with finer granularity could improve the quality of service for the customers allowing them to pay for the actual time they benefit from the service.

The application suffers from the problems due to the current Flash Channels implementation: lack of completely effective economic incentives and parking duration inversely proportional to the payments speed.

It's important to remark that at the moment the Flash Channels Library is not able to provide sub-second throughput because every payment requires some seconds to be accomplished.

It's clear that improvements to the Library to enable really instant payments and the introduction of confidence intervals in transaction timestamps are absolutely necessary to allow Flash Channels to be used in real world use cases.

References

[1] https://medium.com/@sbmeunier/blockchain-technology-a-very-special-kind-of-distributed-database-e63d00781118

[2] https://gendal.me/2016/11/08/on-distributed-databases-and-distributed-ledgers

[3] https://www2.deloitte.com/content/dam/Deloitte/au/Images/infographics/au-deloitte-technology-bitcoin-blockchain-distributed-ledgers-180416.pdf

[4] Andreas M. Antonopoulos, *Mastering Bitcoin. Programming The Open Blockchain*, O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2nd edition, 2017.

[5] Arvin Narayanan, Joseph Bonneu, Edward Felten, Andrew Miller, Steven Goldfeder *Bitcoin And Cryptocurrency Technology: A Comprehensive Introduction*, Princeton University Press, 41 William Street, Princeton, New Jersey 08540, 1st edition, 2016.

[6] https://en.bitcoin.it/wiki/Weaknesses

[7] https://en.bitcoin.it/wiki/Quantum_computing_and_Bitcoin

[8] https://blog.iota.org/a-primer-on-iota-with-presentation-e0a6eb2cc621

[9] https://iota.org/IOTA_Whitepaper.pdf

[10] https://en.wikipedia.org/wiki/Internet_of_things

[11] https://www.reddit.com/r/ethereum/comments/696iln/when_is_ethereum_going_to_run_i

[12] https://iota.readme.io/v1.2.0/docs/the-anatomy-of-a-transaction

[13] https://iota.stackexchange.com/questions/1000/do-transactions-within-the-same-bundle-need-to-reference-specific-branch-and-tru

[14] https://iota.stackexchange.com/questions/1240/what-is-a-transactions-obsoletetag-used-for

[15] https://iota.org/timestamps.pdf

[16] https://iota.stackexchange.com/questions/1242/how-is-ensured-that-a-message-sent-in-a-transaction-is-immutable
https://iota.stackexchange.com/questions/1267/how-to-send-data-payload-and-funds-in-a-secure-immutable-way/1269

[17] https://iota.stackexchange.com/questions/40/what-kind-of-invalid-transactions-are-transported-by-the-network-and-appear-o?rq=1

[18] https://iota.stackexchange.com/questions/1234/what-is-the-initial-own-weight-of-a-transaction-in-iota

[19] https://medium.com/@abmushi/iota-multisig-explained-bca334d250a2

[20] https://blog.iota.org/instant-feeless-flash-channels-88572d9a4385

[21] https://public.tangle.works/winternitz.pdf

[22] https://github.com/iotaledger/iota.flash.js

[23] https://blog.bitmex.com/the-lightning-network/

[24] https://blog.iota.org/iota-development-roadmap-74741f37ed01

[25] https://docs.iota.org/introduction/testnet/introduction

[26] https://www.maize.io/en/content/iota-in-the-new-mobility