Qubic Lite

White Paper v1.0

microhash

August 2018

Contents

1 Motivation

Software technologies have been improving for the last decades at an exponential rate. We now see very promising new development paradigms pop up with increasing frequency. And without a doubt, **distribution and decentralization** of formerly centralized tasks and services is among the most interesting trends in today's world.

Leading the innovative front, distributed ledger technologies such as *Blockchains* and *DAGs* (*Directed Acyclic Graphs*) provide an arsenal of different approaches to do exactly that. However, the major issue each of these technologies has to face sooner or later is **scalability**. A system in which every participant has to be aware of every action in the whole network, will at some point inherently fail to satisfy the ever-growing demand.

Therefore, the foundation for any solution has to be built on a scalable base layer and needs to be utterly dynamic and adaptive itself. One of the most promising projects in this regard is **IOTA** which provides a DAG-based data integrity protocol, called *Tangle* [3], which was designed with scalability in mind from the very beginning. Verification of (data or value) transactions is conducted by the network users themselves. Each participant is only required to verify a tiny sub-graph of the Tangle, instead of the entire global state (as it is required in Blockchain ledgers). This way the user becomes the service provider and there neither is a separation of groups with different interests (see users and miners in Blockchains) nor are there any network fees. This is what makes the IOTA protocol appear as a promising data integrity base layer for distributed applications running on top of it.

The IOTA Foundation has released a theoretical concept of how such a second layer protocol for distributed computing - called **Qubic** (*Quorum Based Computations*) - could be built on top of the Tangle [1]. They hope to have a working alpha ready by the end of 2018 [2].

While the conceptual design of Qubic seems rather promising, the delay until a usable version is released will prevent the community from working with Qubic. Therefore there is a general need for a Qubic-like prototype that can be released much sooner to allow developers to make themselves familiar with the protocol structure early on and start building first applications already. This is exactly what **Qubic Lite** aims to achieve.

2 Qubic Theory

This section summarizes the basic design concepts of the Qubic protocol as envisioned by the IOTA Foundation [1]. They do also apply to Qubic Lite.

2.1 Qubics

The building blocks of Qubic are called **qubics**[1]. They represent a single computational task requested by the outside world and sent to the protocol for execution. Besides some meta data, each qubic contains instructions (the *qubic code*) defining how the calculation shall be performed.

2.2 Oracles

Oracles are (usually) automated network participants which execute the qubic code and publish their results independently. The qubic author (or anybody else who is interested in having the qubic proceed) can incentivize the oracles by paying rewards for correct results. To guarantee decentralization, multiple oracles gather in clusters (*assemblies*). Since oracles are external - potentially malicious - entities, they cannot be trusted individually. Hence a custom consensus algorithm is used to determine which result is considered valid.

2.3 Quorum Consensus

The consensus algorithm used is based on **quorums**. The oracle results are weighted by a certain **voting power** and only if at least two thirds of the weighted results conform

[1] I will use capitalization to distinguish between Qubic (the protocol) and qubics (computations run on this protocol)

to each-other, the quorum is reached and this result is considered consensus. Therefore it would require one third of malicious actors to prevent consensus (34% attack) or two thirds to fake the results (67% attack).

2.4 Epochs

In theory oracles process qubics forever and constantly produce results. This happens in a sequence of consecutive **epochs**, during each of which every oracle can publish a single result. Consensus is then built per epoch from all results that were created in that respective epoch.

2.5 Scalability

Qubic's unique nature enables unique properties. The most significant being **local consensus**: As opposed to traditional Blockchain consensus algorithms, no single Qubic participant has to keep track of the entire protocol events (as required for **global consensus**). Instead each oracle only focuses on the very qubic it processes [2].

This has further advantages since it gives the qubic authors more choices when deploying a new qubic. They can decide how many oracles they wants to have in their assembly to find an appropriate balance between processing fees and decentralization (higher fees → larger assembly → more decentralized).

The fractioning into qubics does not prevent the existence of more complex systems consisting of multiple qubics exchanging information with each-other. In fact, due to discrete epochs predictable data flows between well-defined interfaces are guarantees

[2] As I will show later, the scalability can easily be further extended beyond this with a simple probabilistic method with a game-theoretical foundation, to a point where each oracle only has to observe a fraction of the qubic.

and thus unexpected behaviour that would usually result from putting together code like this can be avoided. Moreover, this architecture allows for enormous yet scalable and dynamic qubic networks to be composed. This opens doors to an **Internet of Computations**.

2.6 Data Integrity

Choosing the Tangle as the foundation for Qubic is not an arbitrary decision. IOTA ensures some very important qualities required by such light-weight distributed computations. Most importantly, the Tangle guarantees data integrity. Data will be stored persistently in a tamper-proof environment: once a transaction has been published, it will be broadcasted to the entire network, buried by other transactions and within seconds it becomes impossible to deny its existence.

3 Qubic Lite Protocol

3.1 IAM Streams

3.1.1 Motivation

Messages sent between entities on the Qubic Lite protocol (oracles, qubics, machines and humans) require authentication. IOTA offers **MAM** (*Masked Authenticated Messaging*) - a messaging layer that is able to provide that among additional privacy features. A MAM streams is a chain of messages, each containing information about how to find and decrypt its successor message.

But due to their linearly linked structure, MAM streams cannot be efficiently traversed to find an element at a certain position in the chain - which is required in order to find the result of a certain epoch in a stream that publishes one result message per epoch. Therefore it appears beneficial to design a messaging layer better suited for use in Qubic Lite.

3.1.2 Idea

IAM (*Indexed Authenticated Messaging*) does not use links between messages. Instead, messages are published to IOTA addresses deterministicly derived from the static IAM stream ID (see 3.1.4) and a message index - a combination of a positive number indicating the position of the message within the stream and optionally a short tryte sequence acting as keyword. In contrast to MAM, IAM does not behave like a chain, but more like a register.

3.1.3 Efficiency

IAM only requires a constant time complexity $O(1)$ (lowest complexity possible) to find the address for a certain index as opposed to MAM which does the same with linear complexity $O(n)$. As the epoch index for a qubic constantly grows over time, this can make quite an increasing difference over time.

3.1.4 Authentication

Authentication is realized with asymmetric signatures. Each stream owns a key pair consisting of a public and a private key [3]. The public key is published in a first transaction, called the **base transaction** of the IAM stream. The transaction hash of the base transaction is the **IAM stream ID**. As it is both the reference to the public key and the starting point for deriving the index addresses, knowing the ID is sufficient to read messages from the stream.

3.1.5 Address Derivation

Addresses are derived from the stream ID, by keeping the first 30 trytes and overwriting the remaining 51 trytes with '9's. The last few trytes are then changed to represent the index in base 27 ('9' in base 27 equals 0 in base 10, 'A' equals 1, 'B' equals 2, 'Z' equals 26, 'A9' equals 27 etc). Example:

```
stream id:  HMSCIRTQEM9GMRENWOXEJYXARGZNUTLFBTSORUSFU9LYTOLTGGYJXCYAJYGEDTDZVUXRUTPMPSWDWO999

index 100:  HMSCIRTQEM9GMRENWOXEJYXARGZNUT99999999999999999999999999999999999999999999999CR
```

[3] As its name suggests, the Winternitz one-time signature scheme utilized for signing value transactions in IOTA is not suited for creating more than one signature. A different scheme has to be used here.

Keywords If a keyword (a short tryte sequence) is used, it overwrites the trytes right after the first 30 trytes. Example for `keyword = "HELLO9WORLD"`:

```
stream id:  HMSCIRTQEM9GMRENWOXEJYXARGZNUTLFBTSORUSFU9LYTOLTGGYJXCYAJYGEDTDZVUXRUTPMPSWDWO999

index 100:  HMSCIRTQEM9GMRENWOXEJYXARGZNUTHELLO9WORLD99999999999999999999999999999999999999CR
```

3.1.6 Reading-Writing Process

The stream owner starts off by publishing a message into the stream. For convention, messages are required to be JSON objects. A signature is created for the object[4]. Then both the message object and signature are combined into an **IAM packet**, the basic unit for information in IAM streams. The packet is then sent through one or multiple IOTA transactions to the address derived for the specific index. Multiple transactions are required if the packet size surpasses the message size of IOTA transactions. In that case the message is split up into fragments which are linked together through the transaction hashes.

The reader checks all transactions on the address derived from the writer's IAM stream at the respective index. For each transaction, the according IAM packet is reconstructed and the signature is checked. If all packets with a valid signature contain the same message, this message is considered valid. If there are different messages conflicting with each-other, they are all invalid and the IAM stream is considered to not have a message at this index. Once there are reliable timestamps on the IOTA protocol, this can be changed to prefering the packet published first.

[4]The index is used as signature nonce to prevent malicious actors from reattaching signed messages to other indexes of the same stream.

3.2 QL-Nodes

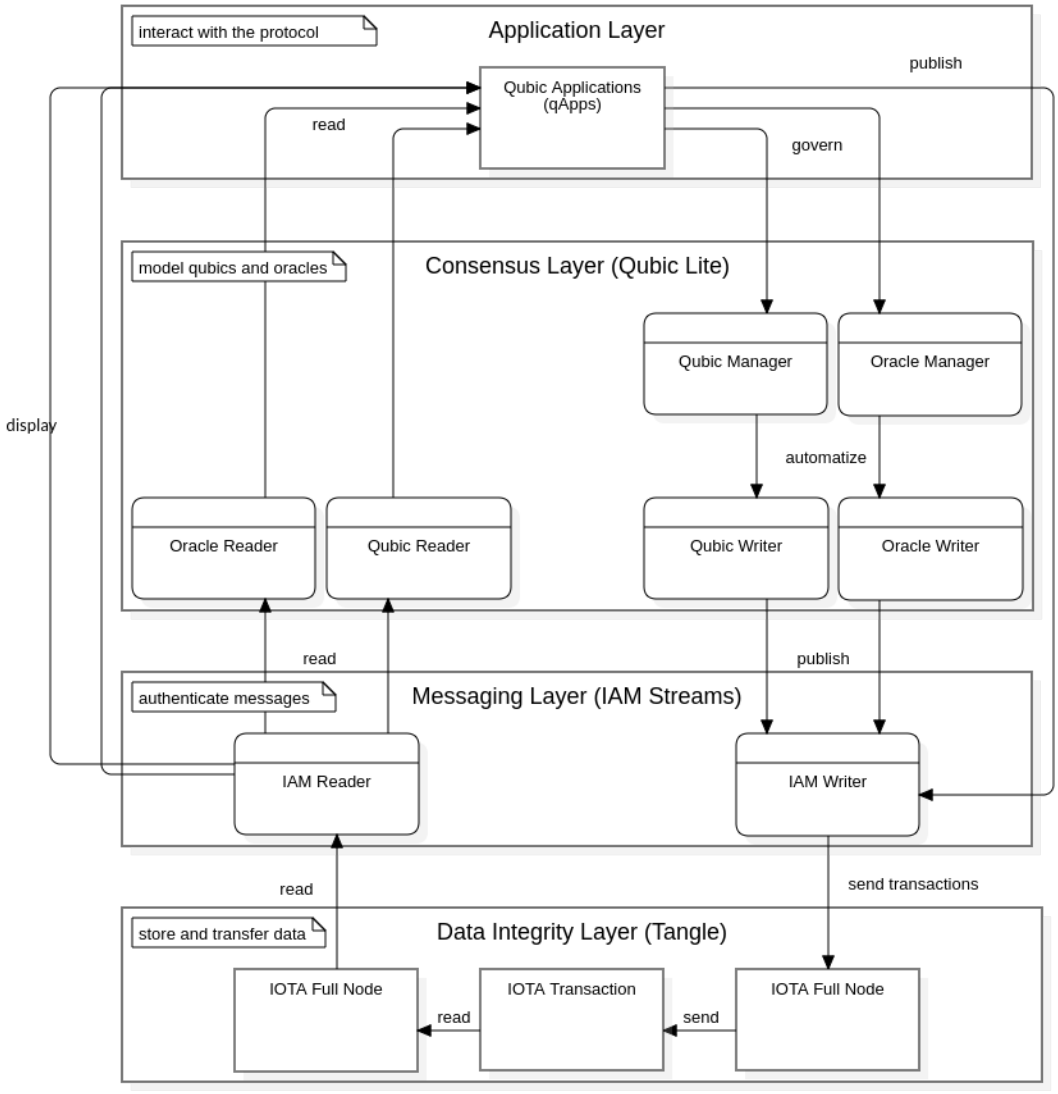

Figure 1: the protocol layering structure

The Qubic Lite protocol can only be accessed through ql-nodes which run the required software to interact with each-other. Communication between ql-nodes happens indirectly via the Tangle. Therefore every ql-node connects to an IOTA full node. As a consequence, ql-nodes are usually anonymous and do share their IP only with that

IOTA node.

Participants do not encounter each-other as ql-nodes, but present themselves as qubics, oracles or IAM streams (which will be introduced in 3.1). For authentication purposes, oracles and qubics utilize IAM streams as a messaging layer to communicate. Each ql-node can simulate an unlimited number of oracles or qubics. It is not possible to link these identities together.

3.3 Qubic Lifetime

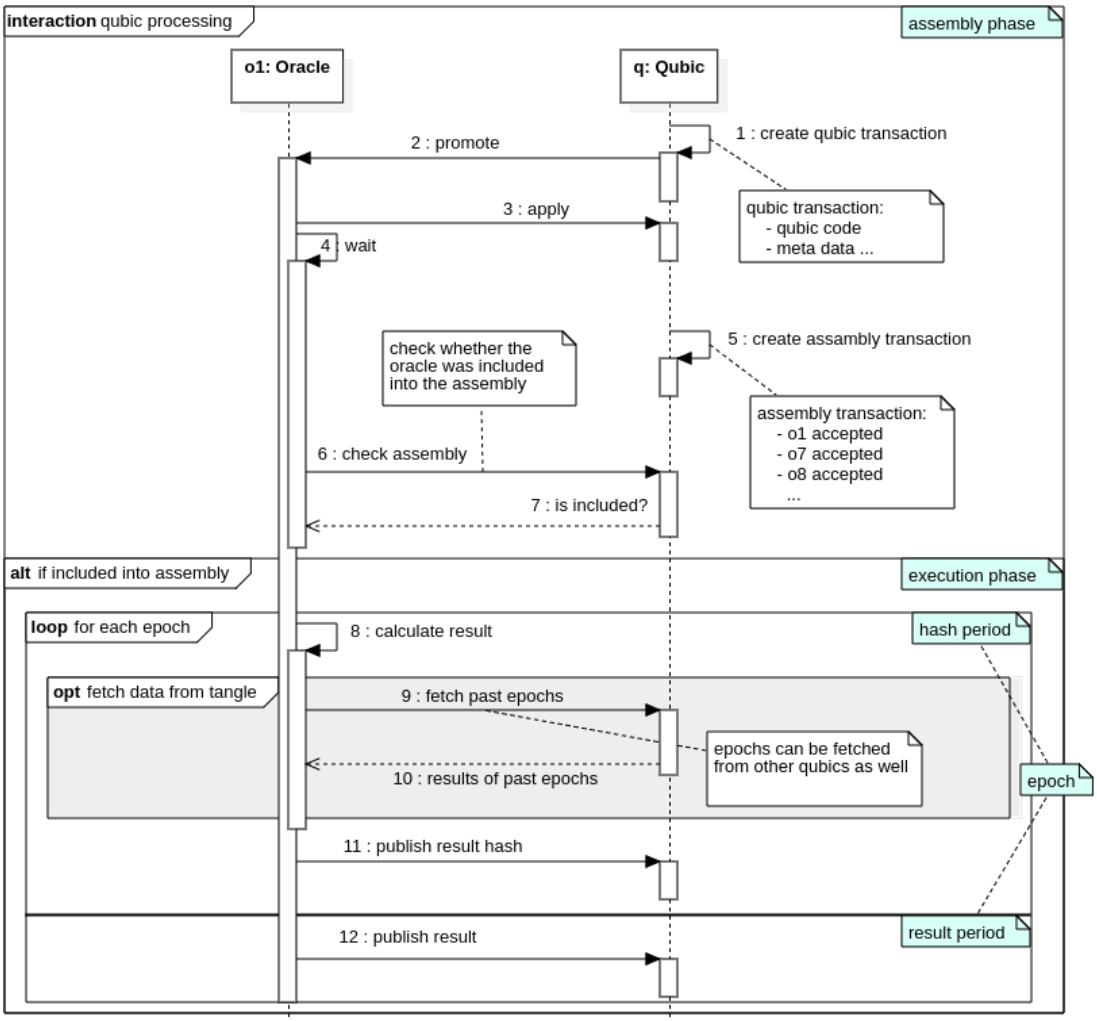

Figure 2: a qubic processed by a single oracle

3.3.1 Assembly Phase

Everything starts with the creation of a new qubic. The qubic author publishes a **qubic transaction** (specified in 3.4) which contains the qubic code and other meta data. After that they promote their qubic publicly on the tangle hoping to find ql-nodes willing to

process it.

Ql-nodes are actively listening for such promotions. Once they find an attractive qubic, they apply as oracles for a position in the assembly. The qubic owner can then select which of the oracles to accept by publishing the list of accepted oracles in a second transaction, the **assembly transaction**. After this, no further action is required from the qubic owner, the qubic has now become an autonomous and decentralized entity.

The qubic transaction contains a timestamp (the *execution start*) marking the end of the **assembly phase** (described in the previous two sections) and the start of the **execution phase**. Just before the execution phase starts, every applied oracle checks the assembly transaction to find out whether it was included into the assembly. Only those included will process the qubic.

3.3.2 Execution Phase

The execution phase is an infinite sequence of epochs starting with epoch 0, followed by epoch 1, epoch 2, etc. Each epoch consists of two consecutive periods: the **hash period** and the **result period**. The duration of each epoch is defined as the sum of *hash period duration* and *result period duration* (see 3.4).

Hash Period During the hash period each oracle calculates the epoch result independently and is then allowed to publish a nonced hash of the result. This hash is required to prevent classroom attacks (oracles copying results from each-other). Depending on the qubic code, processing results from previous epochs of the same qubic or from external qubics is usually required when computing the result. To do this, the oracle reconstructs the consensus of that past epoch. Because several transactions have to be fetched from the tangle for this purpose, it is the most time-consuming part of the result calculation.

Result Period In the result period the oracles make their results public. Results will only be accepted by other assembly members if they, as well as their respective hashes, were published in time and the hash can be validated.

3.4 The Qubic Transaction

The qubic transaction, which is the starting point for everything happening in the Qubic Lite protocol, contains all the meta data that specifies how the qubic has to be handled. It is published at index 0 of the qubic's IAM stream (see 3.1).

Qubic Code The most important data in the qubic transaction is the qubic code. It is a piece of code written in a custom, javascript-like programming language called **QL**[5]. The code will then be executed on the oracle's **QLVM** (*Qubic Lite Virtual Machine*), a safe execution environment within the ql-node software.

Execution Start The execution start is a unix timestamp defining the point of time at which the assembly phase transitions into the execution phase (see 3.3).

Hash/Result Period Duration The hash period duration and the result period duration specify the durations of the respective periods (see 3.3.2) in seconds, as well as the epoch duration, which is simply the sum of both.

Runtime Limit This parameter limits the runtime of the QLVM per qubic epoch. This stops the epoch calculation in case the code enters an infinite loop or requesting data from the Tangle takes too long.

[5]documentation available on `http://qubiclite.org/dev`

4 Project Development

4.1 Current State

QLRI A working version of the **QLRI**[6] (*Qubic Lite Reference Implementation*), the reference ql-node software, has been released. It provides an API for external requests, as well as an inbuilt Web GUI (**Qlite Web**[7]) connecting to the API of the QLRI with the javascript library (`https://github.com/qubiclite/qlite.js`). The QLRI utilizes the Qlite Java library (`https://github.com/qubiclite/qlite.lib.java`) which contains the core logic to interact with the protocol (iam streams, oracles, qubics).

QLVM The provisional QLVM module currently integrated in the QLRI is capable of handling linear sequences of imperative code and is able to process simple data types (strings, integers, doubles) as well as more complex types (arrays, jsons).

PoC "Tangle Farm" (`http://qame.org/tanglefarm`), a first **qApp** (*Qubic Application*) game running on top of Qubic Lite has been built. As a rather complex proof-of-concept it showcases how Qubic Lite can be used as framework to develop decentralized applications without having to explicitly implement the consensus mechanism.

[6] `https://github.com/qubiclite/qlri`
[7] `http://qubiclite.org/qlweb`

4.2 Development Targets

From a technological point of view, Qlite is ready to be used for a wide range of ideas. This is, however, currently prevented by the lack of available resources. Therefore, educating the community about how to work with the software, is the number one priority for the short-term. Besides guides and tutorials, proper documentation of the available libraries and the API is required.

Another target is to make the software run stable and efficiently. Reliability guarantees that everything behaves like expected and therefore avoids confusion and desperation among developers and users. Up until now, the focus was put onto releasing a first working prototype as early as possible. Now that this has been done, quality has become the main priority.

A general rewriting of the QLVM seems necessary, as the current one is highly inefficient in code execution and neither easy to maintain nor extend. Error stack tracing and the definition of functions are missing features. Running the qubic code on an existing language (such as JavaScript) might be the best solution. This implies that a safe environment needs to be built to prevent malicious code harming the oracle owner.

For further adoption, more libraries are requried to attract a larger portion of developers. For non-developers (users) more applications have to be built. Multi-player games might be the best way to reach the community.

5 Glossary

Assembly The cluster of oracles processing a qubic. Consensus is built by determining the quorum among all oracles in the qubic's assembly (see 2.2).

Assembly Phase The first phase during the lifetime of a qubic. It starts with the qubic transaction and ends with the assembly transaction. The purpose of this phase is to allow oracles to apply to join the assembly executing this qubic. (see 3.3.1)

Epoch A time unit of the qubic execution. During each epoch a quorum based result is created for the qubic (see 2.4).

Execution Phase The second phase during the lifetime of a qubic. It starts with the assembly transaction and goes on forever. During this phase oracles publish the results of the respective epochs (see 3.3.2).

IAM *Indexed Authenticated Messaging*. Messaging layer of Qubic Lite. (see 3.1).

Oracle Any entity feeding outside-data into the tangle. In the context of this document oracles execute qubics and publish their calculation results based on which consensus is built (see 2.2).

Ql-node Autonomously acting nodes on the Qubic Lite protocol. Ql-nodes run multiple oracles and qubics parallely (see 3.2).

QL A custom imperative, script language used for writing qubic code (see 3.4).

QLRI *Qubic Lite Reference Implementation*. The official[8] software used to run a ql-node. (see 4.1).

QLVM *Qubic Lite Virtual Machine*, an environment in which oracles execute the code specified in the qubic (see 4.1).

Quorum The consensus algorithm of Qubic. Consensus happens if at least a certain amount of participants agree on a subject (see 2.3).

Qubic Qubic (capitalized) refers to the Qubic protocol as described by the IOTA Foundation [1], while a qubic (lower case) is a computational task sent to this network for execution (see 2.1).

[8] released by `qubiclite.org`

References

[1] IOTA Foundation. Qubic: Quorum-based computations - powered by iota, June 2018. URL: https://qubic.iota.org/.

[2] Klemens Kilic. Interview mit david sønstebø von iota. *Wired*, June 2018. URL: https://www.wired.de/collection/tech/iota-gruender-david-soensteboe-qubic-wird-sich-gegenueber-ethereum-durchsetzen.

[3] Serguei Popov. The tangle. White paper, IOTA Foundation, April 2018. URL: https://assets.ctfassets.net/r1dr6vzfxhev/2t4uxvsIqk0EUau6g2sw0g/45eae33637ca92f85dd9f4a3a218e1ec/iota1_4_3.pdf.