Trustworthy Peer Financing with Ultiledger Blockchain Protocol and Ecosystem

Ultiledger Team[*]

Version 1.0, last updated on July 11, 2018

info@ultiledger.io

Abstract

Long before the birth of banks, insurance companies and other various types of financial institutions, people have started exchanging, trading and financing activities throughout the human history. Financial institutions come at a later phase in the hope of enhancing trust and collaboration between natural entities, however, these institutions have proven to be extremely expensive to maintain, while the extend of trust offered by them have been diminishing for quite a long time and faith in these institutions is dwindling like candles in the wind. The creation of blockchain technologies, especially popularized by Bitcoin, offers a compelling alternative to centralized financial entities as blockchain ledgers are book-kept in a distributed manner and offer tamper-proof solutions. This is a great advancement in both technology and humanity since it brings trust through technology.

Ultiledger is a blockchain protocol and ecosystem that aims for providing state-of-the-art functionalities as well as quality services for peer financing markets. Ultiledger brings improvement to existing blockchain technologies in three major layers: the consensus layer, the storage layer and the smart contract layer. Specifically, introduces a new consensus algorithm which promises high TPS, a hierarchical subchain architecture to further increase scalability, a decentralized storage system for cost-effective data access, as well as a modern smart contract platform for flexible application development. Combined together, these improvements help Ultiledger blockchain achieving high security, high scalability and TPS, and high application flexibility.

1. Introduction

Ultiledger is an open financial protocol that is committed to leveraging blockchain technologies to help any credit organizations including government, banks, corporate companies, individual persons, B2B e-commerce platforms, industrial sectors, *etc.*, to quickly build ultra-high security levels of finance. Its distributed ledger technology (DLT) can achieve ecological zero-cost transaction settlements through a hierarchical setup of blockchains: the combination of a main chain, sub-chains and sub-chain sets. Besides, Ultiledger is able to achieve system security, privacy protection, high efficiency and capital accommodation.

Specifically, Ultiledger focuses on the management and authentication of digital identities, issuance and trading of digital assets, and the integration of IPFS technology into its ecosystem. By means of bringing various files in the peer financing process onto the IPFS network as well as the blockchain,

[*] https://ultiledger.io

Ultiledger is finally brings blockchain technologies into real-world applications while greatly reducing the unnecessary financial and trust costs of intermediate links. Similar to the Bitcoin technology, Ultiledger is a non-intermediate value exchange protocol through the network; but unlike the closed ecosystem of Bitcoin, the Ultiledger protocol comes with an exchange functionality, which can be interconnected with external financial institutions. We strive to build a safer platform with more open design, lower costs and at the same time superior user experience. Last but not least, Ultiledger is extremely responsive to various regulations from different governments and takes advantage of blockchain as a technology enabler for a wide range of industrial entities.

Ultiledger aims to build a fast, low-cost, efficient, secure, and reliable blockchain economic ecosystem that meets large-scale daily business needs. Ultiledger brings improvement to existing blockchain technologies in three major layers: the consensus layer, the storage layer and the smart contract layer. Specifically, introduces a new consensus algorithm which promises high TPS, a hierarchical subchain architecture to further increase scalability, a decentralized storage system for cost-effective data access, as well as a modern smart contract platform for flexible application development. Combined together, these improvements help Ultiledger blockchain achieving high security, high scalability and TPS, and high application flexibility.

In the remainder of this whitepaper, we first survey the relevant research work. Then we introduce how consensus is reached in Ultiledger, followed by its decentralized storage architecture and smart contract platform. Then we list a number of additional technical aspects worth discussing before we finally conclude this whitepaper.

2. Related Work

It is widely accepted amongst blockchain researchers and practitioners that the community's huge interests have been ignited from the inception of Bitcoin (Nakamoto, 2008). Traditionally, when people make payment transactions, they have to rely on and trust a third-party, who acts as an intermediary to verify the validity of such a transaction before settle down the payments. Bitcoin has the same capability as traditional payment intermediaries, except that it is not a centralized entity but rather a decentralized system where transaction participants do not need to rely on any single intermediary since such parties are often doubted whether they can be exploited to cheat its users. Bitcoin's solution to the trust problem is using multiple independent entities to verify transactions and altogether bookkeep a single tamper-proof public ledger.

The key contribution of the Bitcoin blockchain is its consensus algorithm, which decides how agreement is reached on which participating node (among all nodes in the verifying network) is able to append a new block to the public ledger. Bitcoin adopts an approach named Proof-of-Work (PoW), where nodes are only granted rights to broadcast their blocks when they have performed a lot of effort evaluated on their computing power, by finding a nonce n such that the hash of the block satisfies a thresholding condition $\mathcal{T}$ regulated by the network difficulty which is updated every 2016 blocks as predefined in Bitcoin's implementation.

Besides the rapid development and wide acceptance of Bitcoin during the past ten years, a variety of blockchain architectures have been proposed, such as Litecoin[2], Ethereum[3], Ripple[4] and Stellar[5]. As PoW algorithms generally have a huge energy footprint, more and more blockchain architectures are switching to more efficient consensus algorithms, such as Proof of Stake (PoS), Byzantine agreement such as PBFT (Castro and Liskov, 2002), tendermint (Kwon, 2014) based on PoS membership mechanism and Permissioned (Androulaki et al., 2018) architecture.

Besides the energy inefficiency issue with PoW algorithms, PoW has also been criticized as being unfair: miners with modern and expensive ASIC equipment could find the desired nonce faster than those with poorer hardware setup. Proof-of-Stake (PoS) is supposed to mitigate this inequality by determining the change of a miner getting to mine a new block by the amount of stake this miner possesses: the more stake a miner owns, the higher probability he has to mine a new block. Specifically in the vanilla version (Nguyen and Kim, 2018) of PoS, if there are a total t staking units

[2] https://litecoin.org/
[3] https://ethereum.org/
[4] https://ripple.com/
[5] https://www.stellar.org/

from all the miners, and miner $\mathcal{M}$ owns a specific amount of stake s ($s < t$), the probability for $\mathcal{M}$ getting the right to mine a new block is $P_{\mathcal{M}} = \frac{s}{t}$. In addition, in PoS it requires any attackers to own more than 50% of all stakes in the network to perform a double spending attack. Since the more stake a network node possesses, the more likely it is willing to maintain the smooth operation of PoS validation process, it is then theoretically difficult and illogical to perform double spending attacks. However, PoS has also its own drawbacks and it is often criticized as not resilient to *nothing at stake* attacks, where no penalties are enforced when a validation node votes for multiple competing chains during a fork.

Many early PoS protocols have suffered from the *nothing at stake* issue, including PeerCoin (King and Nadal). To mitigate this challenge, later protocols such as the Casper project of Ethereum (Buterin and Griffith, 2017) introduces a penalty mechanism replicated from the economics of PoW, where there can be an implicit penalty voting for the wrong chain, while Ouroboros (Kiayias et al., 2017) of Cardano[6] solves this issue by means of a slot assignment and checkpointing mechanism. Another PoS variant is the Delegated Proof-of-Stake (DPoS) (Larimer, 2014), where network nodes with smaller stakes may team up to increase their chance of mining a new block, thereby helping balancing out the power of large stakeholders. Tendermint (Kwon, 2014) is another important consensus algorithm, in the sense that it combines two different worlds, *i.e.*, the BFT and PoS, so that it brings originally permissioned BFT into a permissionless chain setting.

Besides the variety of chains in the permissionless domain, there remains a huge demand for permissioned blockchain architecture. Contrary to public or permissionless blockchains where each node can participate in transaction validation, permissioned or consortium blockchain systems give privilege to a specific amount of nodes over the validating rights, where the rest of participating nodes may still participate in transaction validation after the aforementioned privileged nodes have reached consensus. The demand for permissioned blockchains generally come from practical considerations, for instance, while auditing is important to large financial institutions to be considered as regulation compliant, it has proven to be infeasible to conduct auditing on public nodes who may hop on and off a network during any given specific time period. As a result, it is also important to consider traditionally permissioned setups. Ripple is one of the most prominent representatives which runs a permissioned blockchain. Ripple determines who may act as transaction validator on their network while at the same time setting up its own nodes in different locations around the world.

In addition to the auditing and governance capability, permissioned blockchains generally come with a number of other advantages compared to permissionless ones. First of all, they are faster and more energy efficient since they generally do not require the resource-consuming PoW process. Moreover, permissioned and easily implementable compared to permissionless blockchains. Last but not least, permissioned blockchains have so far the most real-world applications and are supported by a large number of traditional business entities and modern ones alike.

There are a few consensus algorithms under the category of permissioned consensus algorithms besides Ripple, including Stellar (Mazieres, 2015) and Hyperledger Fabric (Androulaki et al., 2018). Stellar introduces a concept of quorum slices. A quorum is a set of nodes that act together reach consensus while a quorum slice is its subset designed to help a node in reaching consensus. The Stellar Consensus Protocol (SCP) consists of a nomination protocol and ballot protocol. SCP provides decentralized control without a central authority, asymptotic security, flexible trust and low latency. Initially the nomination protocol is run in order to propose transactional details for further consensus. Each node receiving such proposals will vote for a single one and eventually results in unanimously selected proposals for that slot. After successful execution of the nomination protocol, the nodes deploy the ballot protocol, where the federated voting happens to decide either to commit or to abort the proposal resulted from the nomination protocol. This ballot protocol results in externalizing and finalizing results for the current slot and the discarded ballots are marked as irrelevant. SCP also assures avoidance and management of stuck states and thus provides liveliness. However, it does not guarantee security if a node chooses an inefficient quorum slice. Hyperledger Fabric follows the PBFT (Castro and Liskov, 2002) scheme and it defines two different type of nodes: *validating nodes* who run the consensus algorithm and thus validate the transactions, and *non-validating nodes* who act as a proxy to connect clients to validating peers but are not capable of executing any transactions but can verify them.

[6] https://www.cardano.org/

3. Consensus Algorithm

A blockchain is a type of distributively replicated state machine that resembles the form of a data chain, where later data blocks refers to a single ancestor block often identified by its hash. When a blockchain grows, new blocks are included in the state machine and the propagated to all participating nodes within the network so that every node in the network has a single global view of all transaction data.

When multiple nodes intend to append data to the same data store, a consensus is need to decide which node has the privilege of conducting so. Thus a consensus protocol is required for coming into an agreement on how the data modification is handled. Note that consensus algorithms are not limited to the applications of blockchain. As a matter of fact, any computer algorithm that relies on multiple processes maintaining a common state of data relies on solving the consensus problem. In this section, we detail the design of Ultiledger consensus algorithm. First prepare the reader with the relevant background, then we discuss about the consensus algorithm with mathematical proofs on its security and efficiency.

3.1. Background

In any decentralized data store, it is favorable to have the following capabilities:

1. Consistency: each read request results in either the most recent data or an error;
2. Availability: each read request receives a non-error copy of the data without guarantee that it is the most recent version;
3. Partition tolerance: the data store continues to function despite an arbitrary number of messages are dropped or delayed.

However, among the above capabilities, it is proven that it is impossible for a distributed data store to simultaneously provide more than two. This is known as the CAP theorem (Gilbert and Lynch, 2002):

Theorem 1. *It is impossible for a distributed data store to simultaneously provide more than two out of the following three guarantees: consistency, availability and partition tolerance.*

In particular, given the fact that blockchains are inherently partition tolerance, the CAP theorem implies that one has to choose between consistency and availability. While the blockchain is an immutable chain of transaction data, it is possible for each node to build up a different chain also known as a fork, which is solved by the consensus algorithm. In this case, the blockchain data store reaches eventual consistency, *i.e.*, *finality*.

In order to ensure the high availability of Ultiledger , it is thus mandatory that the blockchain consensus algorithm helps establish an *eventually consistent* blockchain.

3.2. Federated Byzantine Agreement

Due to the high demand of financial institution's compliance capabilities, consensus algorithms with permissioned architecture are popular in the financial sector because such algorithms can provide a good auditing process and capability. Since Ultiledger 's customers are mostly financial institutions and companies and individuals that are closely linked to financial institutions, we have chosen the permissioned algorithm, which is built on Federated Byzantine Agreement (FBA) and provides a series of security as well as functionality improvements.

A permissionless blockchain is open to anyone, as a result, any node may join the network and remain anonymous. It is thus possible and sometimes incentive for a node to tamper transactions and include them in a new data block. Therefore, it is highly possible that the blockchain can end up in a fork, where one chain contains the tampered transaction while the other contains only valid transactions.

The ultimate goal of a consensus algorithm is to avoid these forks, so that every node agrees to a single version of the actual transactions and only maintains a global copy of the blockchain. On the other hand, although in a permissioned blockchain the participating nodes are known and chosen, consensus is still required because the trustworthiness of the participating nodes are questionable either due to faulty setup or unreliable communication channels.

Federated Byzantine Agreement (FBA) is a model for consensus using nodes, quorum slices and quorums. Simply put, in FBA each participant knows of others (quorum slices) it considers trustworthy. It waits for the vast majority of quorum slices to agree on any slot contents before considering the transaction settled. In turn, these quorum slices do not agree to the transaction until the participants they consider trustworthy agree as well, and so on. Eventually, when enough of the network nodes accept the slot contents, they are externalized so that no attackers may roll back such slot contents. Only then the transaction is considered to be settled. Compare to Byzantine agreement systems, nodes that involve in the transaction decide which quorum slide they trust in a decentralized manner.

During the execution of a specific consensus algorithm in FBA, each node declares the input and broadcasts it after receiving the input $\mathcal{S}$. When a node detects that its trusted quorum slices have declared receipt or acceptance of $\mathcal{S}$, it broadcasts its acceptance status. This process is repeated until all nodes accept $\mathcal{S}$ and agree.

3.3. Permissioned System with Decentralized Authority

During the initial setup of an FBA, new nodes that wishing to join the network has to decide by themselves which quorum slice they choose to trust. Furthermore, they do not have to be known and verified before joining the network. As a result, FBA membership is open and the control is decentralized so that nodes can choose whom they trust. Network-wide quorums emerge from decisions made by individual nodes.

However, due to lack of central authority to enforce node verification, it can potentially take a long time before a new node proves itself trustworthy in the network. In addition, when a large number of new nodes intend to join the network at a specific time period, node admission negotiations may briefly overwhelm network traffic and disrupt the normal slot content validation.

In order to mitigate such issues, a decentralized authority mechanism is desirable. Proof of Authority (PoA) is a family of consensus algorithms for permissioned blockchain whose prominence is due to performance increases with respect to typical BFT algorithms; this results from lighter message exchanges.

Note that PoA mentioned in this technical paper refers to the mechanism of introducing trustworthiness with newly joined network nodes, it is not to be confused with Proof of Assets, which is an application layer built on top of Ultiledger blockchain with the aim of providing account certificates through the nature of public blockchain systems.

PoA algorithms rely on a set of $\mathcal{N}$ trusted nodes called the authorities. Each authority is identified by a unique id and a majority of them is assumed honest, namely at least $\mathcal{N}/2+1$. The authorities run a consensus to order the transactions issued by clients. Consensus in PoA algorithms relies on a slot validation rotation schema in order to fairly distribute the responsibility of block creation amongst authorities.

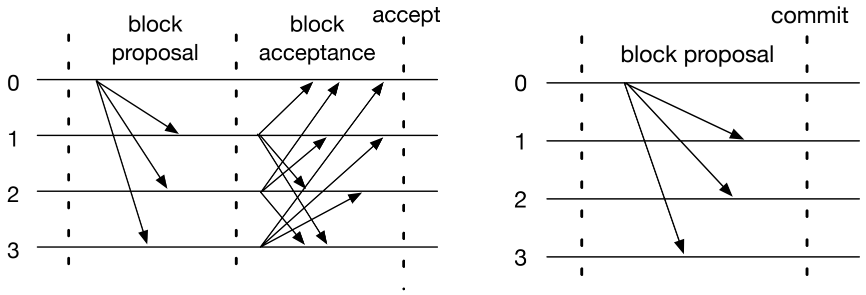

Figure 1: Message patterns of different PoA algorithms.

There are currently a range of PoA algorithms to achieve decentralized authority, namely Aura and Clique. Different PoA algorithms take advantage of slightly different message patterns, as shown in Figure 1. In order to enable light message exchange processes, Ultiledger opts for Clique, which represents a class of PoA consensus with minimal messaging.

3.4. Gossiping Federated Byzantine Generals

In an FBA with $\mathcal{N}$ nodes, there is a theoretically upper limit of $2^{\mathcal{N}}$ quorums and $2^{2^{\mathcal{N}}}$ quorum slices. When communicating slot contents, each node has to select a number of nodes from a extremely wide range of quorum slices to communicate with. Such communication can be very slow, thus leading to slower network consensus and in turn lower TPS. In order to target for higher TPS, *e.g.*, more than 10k TPS, we rely on a more efficient network dissemination protocol named *Gossiping* (Boyd et al., 2006; Van Renesse et al., 2008).

Specifically, the gossip-based data dissemination protocol offers three primary functions:

1. Membership management by continuously identifying available member nodes as well as removing inactive nodes from the network;

2. Bulk transmission of existing ledger data to newly connected nodes in order to allow maximum state machine replication;

3. Slot contents proposal as well as quorum slice communication to quickly reach consensus and externalize proposals at maximum speed; Cross chain ledger data mirroring so that data are kept up to date between the main-chain and its sub-chains (details will follow in the next section).

Adoption of gossip protocols offers a range of advantages to the Ultiledger blockchain, including:

1. Fault-tolerance: the gossip protocol have proven to operate in networks with irregular connectivity. And they work well in such situations because a node shares the same piece of information several times to different nodes. As a result, if a node is temporarily not accessible, the information is propagated through a different node and route. In other words, there are a great number of routes via which information can be propagated from a source to its destinations.

2. Scalability: the gossip protocol is scalable because generally it takes $\mathcal{O}(logN)$ rounds to reach all nodes, where N is number of nodes. Also each node sends only a fixed number of messages independent of the number of nodes in the network. Furthermore, a node does not wait for acknowledgments, and it does not take any recovery action if an acknowledgment is missing. Such a system can easily scale to millions of nodes.

3. Convergent consistency: the gossip protocol achieve exponentially rapid spread of information and thus converges exponentially to a globally consistent state after a new event occurs.

4. Extreme decentralization: gossip offers an extremely decentralized form of information propagation with acceptable latencies.

5. Robustness: nodes play a more or less equal role in the network, as a result, a failed node will not prevent other nodes from sending and receiving messages. Particularly, each node can join or leave whenever it pleases without seriously disrupting the system's overall quality of service.

Network nodes leverage gossip to exchange ledger and channel data in a scalable manner. Gossip-based broadcasting operates by nodes receiving messages from other nodes on the channel, and then forwarding these messages to a number of randomly-selected peer nodes on the channel, where this number is a configurable constant. Nodes can also exercise a pull mechanism, rather than waiting for delivery of a message. This cycle repeats, with the result of channel membership, ledger and state information continually being kept current and in sync. For dissemination of new blocks, the leader peer on the channel pulls the data from the ordering service and initiates gossip dissemination to peer nodes.

3.4.1 Mathematical Formulation of GFBG

The node topology in the Ultiledger network is represented as $G = (V, E)$, where V is the node set and E is the connection information between the nodes. For simplicity, we denote $n = |V|$ and $m = |E|$. That is, there are n network nodes interconnected with a total of m edges.

We denote the set of neighbors of node v as $N(v) = \{u \in V \mid (uv) \in E\}$, then the degree of v $d(v) = N(v)$. Initially, the validation leader node pertains a piece of slot content, whose size is b bits. And the goal is to propagate these b bits of information throughout the network G.

When the gossiping process starts, it is run in multiple rounds. Within each round, each node v randomly selects a single node $u \in N(v)$ from its neighbors and propagate the message to $u \in V \backslash v$. In particular, Ultiledger adapts to a general randomized gossip algorithm, namely *uniform gossip*, where the destination of message propagation is selected uniformly at random (*u.a.r.*) according to uniform distribution $P(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$.

Generally, three basic operating modes are available to deliver information between two message propagation entities: `push`, `pull` and `push+pull`. In `push` v sends a message to u, in `pull` a message is only transferred from u to v and in `push+pull` the communication between v and u are bidirectional. The actual messages can contain only the message itself or additional meta information such as state description and counters. In order to support parallelization, a pointer jumping operation is considered, where the destination of the next round pointer is the pointer at which the current pointer points to.

In Ultiledger network there are two type of nodes, *i.e.*, leaders and proxies, and the message propagation algorithm runs as follows in five consecutive phases.

1. Each node performs a `push` operation in each step of this phase. At the end of this phase there will be $log^2(n)$ nodes pertaining the propagated message after $\mathcal{O}(log(log(n))$ steps.
2. Each node performs a coin-flipping procedure in order to decide whether it becomes a leader or proxy, with a corresponding probability of $\frac{1}{2\sqrt{log(n)}}$ and $1 - \frac{1}{2\sqrt{log(n)}}$.
3. Each proxy node chooses leaders by conducting five pointer jumping procedures for $c\sqrt{log(n)}$ rounds each, so that $n - \mathcal{O}(n)$ nodes are aware of two leaders with high probability. Note that each proxy keeps track of exactly two leaders to reach a tradeoff between cost and efficiency.
4. In every round, each proxy node communicates randomly with a node from its list of leaders. When a connector receives the message, it only repropagates it once in the next round with the `push` operation to its second leader. The leader nodes send the message in all channels with a `pull` operation. This phase will last $c\sqrt{log(n)}$ rounds.
5. Every node v performs the normal `push+pull` operation using the median counter algorithm $ctr(v, b)$ for $c\sqrt{log(n)}$ rounds, so that v is aware of the propagated b bits of information and $ctr(v, b) = 1$.

Especially, in the median counter algorithm (Karp et al., 2000), each node can be in one of the four states $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$ and $\mathcal{D}$. Specifically, State $\mathcal{B}$ is further subdivided in substates $\mathcal{B}_1$, through $\mathcal{B}_{cmax}$, where $cmax = \mathcal{O}(log(log(n)))$ and $cmax \in \mathbb{Z}$. Initially, all propagated nodes in Ultiledger network are in state $\mathcal{B}_1$ or otherwise in state $\mathcal{A}$.

The state transfer is specified as follows:

- If a node v in state $\mathcal{A}$ receives message b only from nodes in state $\mathcal{B}$, then it switches to state $\mathcal{B}_1$.
- If v obtains message b from a node in state $\mathcal{C}$, then it also switches to state $\mathcal{C}$.
- If v in state $\mathcal{B}_i$ communicates with more nodes in state $\mathcal{B}_j$ $(j \geq i)$ than with nodes in state $\mathcal{A}$ or $\mathcal{B}_k$ $(k < i)$, then v switches to state $\mathcal{B}_{i+1}$.
- If v gets b from a state $\mathcal{C}$ node, then it switches to state $\mathcal{C}$.
- If v is in $\mathcal{C}$ state and has sent b for $cmax$ times, then it switches to state $\mathcal{D}$ and stops propagating the message.

3.4.2 Efficiency of GFBG

Based on the mathematical formulation and background for the GFBG propagation algorithm, we have the following theorem regarding its efficiency:

Theorem 2. *At the end of each iteration of the GFBG algorithm, at most $\mathcal{O}(R)$ nodes have not been propagated with b bits of the original slot contents, while all others have been properly propagated with high probability. Here R is the number of remaining nodes to be propagated about the message. The algorithm has a computation complexity of $\mathcal{O}(\sqrt{log(n)})$ and transmits in total $\mathcal{O}(n(log^{\frac{3}{2}}(n) + b \cdot log(log(n))))$ bits of messages.*

We prove Theorem 2 in the remainder of this section. First we reproduce Lemma 3 below, which is proven by Avin and Elsässer (2013):

Lemma 3. *After the fourth phase the number of propagated nodes is at least $\frac{n}{2^{7\sqrt{log(n)}}}$ with high probability.*

Let $\mathcal{P}(t_0)$ be the set of propagated nodes at the end of the fourth phase. Then obviously the total communication cost is $\mathcal{O}(n \cdot b)$ in the fourth phase, since each node transmits at most twice the message, and the number of leaders is bounded by $\mathcal{O}(\frac{n}{2^{\sqrt{log(n)}}})$.

Obviously, as long as the condition $|\mathcal{P}(i)| \leq \frac{n}{log^2(n)}$ holds, it will always be true that $|\mathcal{P}(i+1)| > (1+\epsilon) \cdot |\mathcal{P}(i)|$ ($\epsilon > 0$ and ϵ is a constant).

Such a condition holds since at the end of the fourth stage we only have nodes in $\mathcal{B}_1$ or $\mathcal{A}$, while the original median counter algorithm may contain state $\mathcal{B}_j$ and $\mathcal{C}$ nodes at the same time step, where $j > 1$. As a result, such nodes will stop earlier propagating the message through the Ultiledger network. When $|\mathcal{P}(i)| \geq \frac{n}{log^2(n)}$ for the first time, the communication cost is bounded by $\mathcal{O}(n \cdot b)$ bits.

Once the message is distributed to $\frac{n}{log^2(n)}$ nodes, one needs $\mathcal{O}(log(log(n)))$ additional steps to disseminate the message to every $v \in V$, therefore the total cost is bounded by $O(b \cdot n \cdot log(log(n)))$ bits. To sum up, the communication throughout the Ultiledger network in GFBG algorithm is upper bounded by $\mathcal{O}(n(log^{\frac{3}{2}}(n) + b \cdot log(log(n))))$ bits, where $\sqrt{log(n)}$ is the number of steps in the third phase, while the $log(n)$ is the size of the message in bits in polynomial relation to n.

3.4.3 Summary of GFBG

In the Ultiledger network, each node communicates with its nodes in the quorum slice in a random manner. After some seemingly chaotic communication, the status of all nodes in the final quorum slice will be agreed. The Gossip algorithm assumes that a node may know all other nodes, or only a few neighbors, but as long as these sections can be connected through the network (instead of a *bipartite network*, the algorithm can guarantee that their states are consistent.

Even if there are disconnected nodes in the network or in the cases of new nodes join, after a period of time, the status of these nodes will be consistent with other nodes. Therefore, Gossip is a redundant, fault-tolerant, eventually-consistent algorithm with natural distributed fault tolerance and the ability to optimize the speed of nodes in the network. With gossip communication Ultiledger can achieve lower communication overhead and at the same time higher TPS.

3.5. Hierarchical Chain Architecture

Another improvement of Ultiledger blockchain's TPS lies in its hierarchical chain architecture. That is, Ultiledger consists of a single main chain and a set of sub-chains powered by the FBA consensus. Compared with existing blockchain architectures that aim to create a single blockchain with global transaction ledgers, Ultiledger permits a set of blockchains to run in parallel with one another while retaining interoperability.

On the top of the hierarchy lies the Ultiledger main chain, which manages many semi-independent sub-chains. The sub-chain setup is to some extend similar to database sharding technologies, which is a well-known database scaling technique. Generally, the sub-chains run independently from the main-chain, with the exception of its creation and merging back to the main-chain.

After sub-chains are merged, the final outcome of these sub-chains are committed to the main-chain, in order to allow the man-chain to keep up to date. Similarly, when a sub-chain is created, the relevant information are mirrored from the main-chain to corresponding sub-chains in order for the sub-chains to keep up with the state of the main-chain. In order to ensure data consistency, data communicated across different sub-chains are conducted by posting Merkle-proofs as evidence that the information has been successfully sent and received.

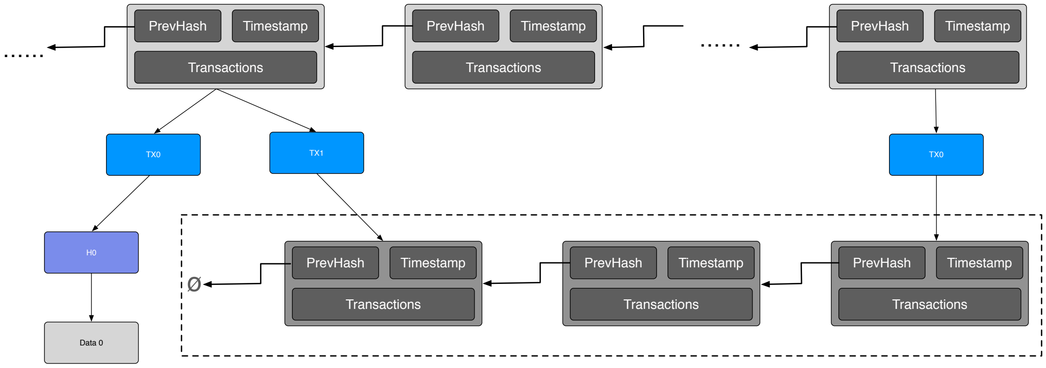

Figure 2: Generating a subchain and merging back to the parent chain.

The main-chain in Ultiledger is a blockchain that hosts a distributed ledger that is synchronized throughout the network. It also contains the issued tokens, which can be held by individual users or more generally different accounts. These tokens can be moved from the main-chain to sub-chains. The main-chain also ensures that global supply of the token is kept constant or fluctuate according to that of the network.

The hierarchical blockchain structure serves as a great instrument of separation of concerns between different organizations' internal workflow and the public ledger where all data are transparent. In subchain systems hosted within individual organization's internal network, this organization is responsible for its own internal operations and such operations are invisible from the top-level blockchain.

As shown in Figure 2, a subchain always derives from a parent chain, carrying over the necessary data as well as tokens. Then as a subchain, participants may conduct transactions independently to its parent chain. Once the required workflow is completed, another transaction is made, so that the final output data as well as tokens are transferred back to the parent chain without exposing every transaction completed on the subchain.

3.5.1 Consensus in Subchain

Subchains are a great approach for improving the scalability of a blockchain, since such an architecture unleashes the full power of parallelization. In order to further increase the TPS of Ultiledger blockchain, we can take advantage of different type of consensus algorithm within the subchain.

More specifically, nodes within the subchain generally have fewer nodes (often magnitudes lower), and the nodes in subchain are not as heterogeneous or dynamic as those in the main chain. Based on these observations, we propose to take advantage of the Raft consensus algorithm (Ongaro and Ousterhout, 2014) in subchains. Raft has the following characteristics that make it a fit into subchains:

1. Strong leadership: Raft consensus protocol assumes a stronger form of leadership than other algorithms. Such a decision greatly simplifies the management of the message propagation process.

2. Leader election mechanism: Raft uses randomized timers to decide which node becomes the leader, which adds very little communication overhead except the heartbeats, which are already a prerequisite in the majority of consensus algorithms. Such a mechanism also enables rapid conflict resolving.

3. Membership updates: it is fairly easy to replace a set of nodes in a cluster with a new set, provided that the majority nodes in both clusters intersect. This ensures seamless updates without disrupting the consensus algorithm execution and resulting in high scalability.

Raft allows the leader to take care of all communication with clients of the application and needs to be contacted directly by them, thus the system is only available when a leader has been elected and is alive. Although it may seem a strong assumption, practical deployment results suggest that Raft can indeed result in scalable consensus in less dynamic environments, which suits perfectly for subchains in Ultiledger.

3.5.2 Communication Across Chains

It is often desirable that entities in different chains are able to communicate and exchange data. In Ultiledger blockchain, communication across chains are easy to implement thanks to its hierarchical chain structure. In fact, we provide two different approaches to support interchain communication, *i.e.*, proxying through the main chain and creating another subchain.

Figure 3 illustrates how interchain communication is conducted using the main parent chain as an intermediary. In this case, all communication goes through the main chain and thus the exchanged data are recorded in the public blockchain. This is desirable when transparent proofs of exchange is required.

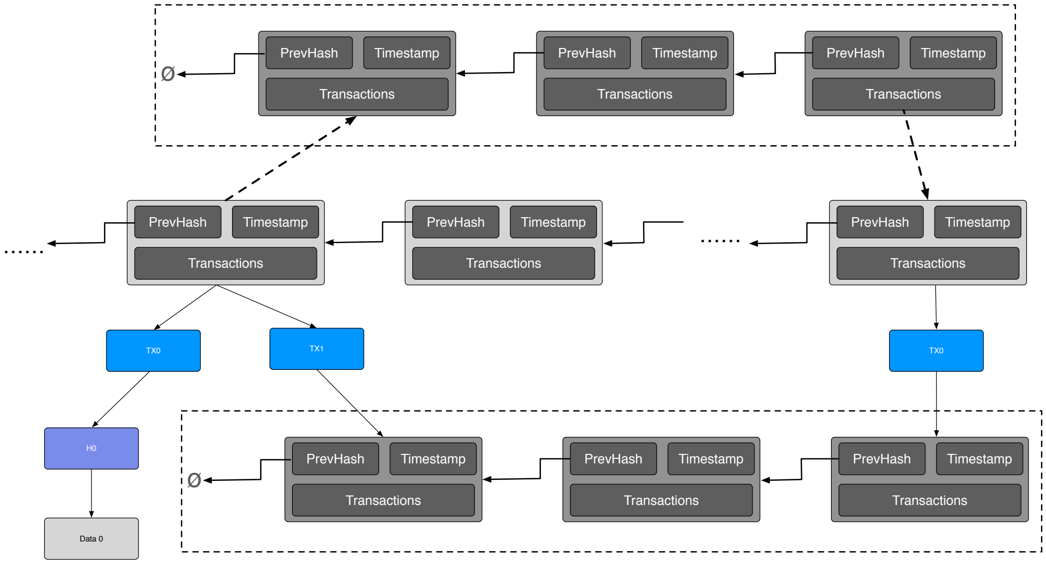

Figure 3: Communication between subchains by proxy of the main chain.

Figure 4 illustrates the interchain communication between two subchains without any intermediary, instead, a new subchain is created from the current subchain. After transactions in the new subchain are completed, the final output are transmitted back to its parent chain, thus finalizing the interchain communication.

4. Decentralized Storage

Blockchain as a pure distributed data store is generally not ideal for storage of massive data, since all data in the blockchain are replicated across all nodes. This is obviously not a scalable approach for industrial level applications, where data are being generated in units of terabytes and petabytes every day. In order to make the Ultiledger practical for enterprise level applications, we devise an decentralized storage architecture that i) introduces different type of storage nodes for assisting more flexible business scenarios and ii) connects to a global storage service, namely IPFS, for business data storage and reference the relevant file hashes within the blockchain.

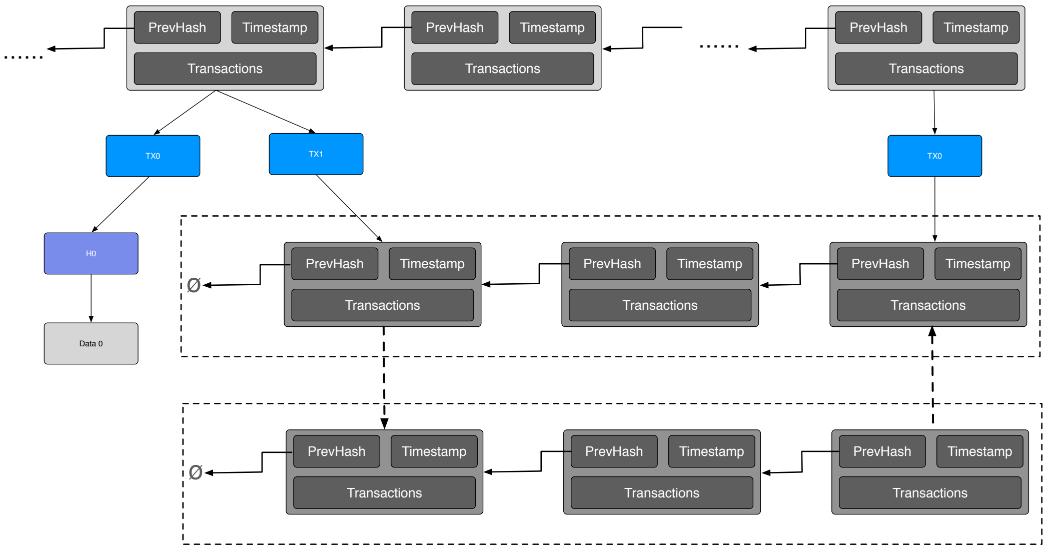

Figure 4: Communication between subchains by creating another subchain.

4.1. Global Storage

IPFS (Benet, 2014; Dias and Benet, 2016) is a promising technology stack that is built on top of a range of well-tested technologies, including Distributed Hash Tables (DHTs), the Git version control system as well as BitTorrent for data synchronization. It establishes a P2P swarm that allows the exchange of files known as IPFS objects. The totality of IPFS objects forms a cryptographically authenticated data structure known as a Merkle DAG (as shown in Figure 5) and this data structure can be used to model many other data structures.

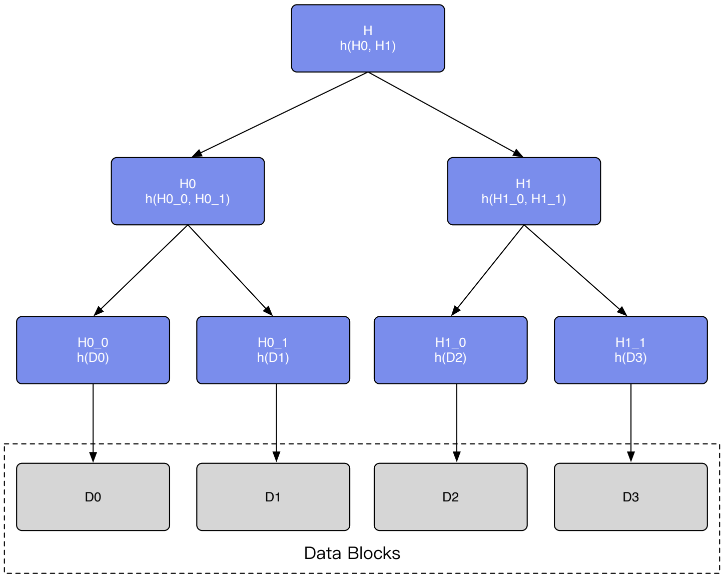

Figure 5: Decentralized global data storage.

As a distributed storage system, IPFS is designed for retrieving and sharing IPFS objects, which is a data structure containing two parts: the data and the links. The data in an IPFS object is a blob of unstructured binary data with a maximum size of 256 kB, and the links are essentially an array of link structures to other IPFS objects. More specifically, a Link structure has three sub-fields, *i.e.*, the name of the link, the hash of the linked IPFS object and the cumulative size of the linked IPFS object, including objects following its links. The size sub-field is mainly used for internal optimization, especially for optimizing the performance of P2P networking.

External applications can easily refer to an IPFS object, which is accessible with the object's hash that is encoded in Base58 format. In the case of Ultiledger blockchain, it is extremely beneficial to delegate data storage to IPFS as much as possible, so that a large number of transactions may fit into a single block. As a consequence, such a decentralized storage also helps improving the TPS of Ultiledger.

Figure 6 illustrates an overview of how the blockchain storage interoperates with the IPFS network. As shown, transactions in Ultiledger are essentially the Merkle root of a larger number of data references, where the data may be easily accessed by traversing the Merkle DAG.

Thanks to this design, Ultiledger bears a number of desirable advantages, which we summarize below:

1. Content deduplication: same data objects lead to the same hash in the IPFS network, thus it is no longer necessary to store the same piece of information multiple times within the same blockchain. Instead, each data object is uniquely accessible via its hash fingerprint.
2. Tamper-proof: the adoption of hash fingerprinting as well as the construction of Merkle DAG ensures that any data object that has been tampered will result in a totally different access address, thus making the data objects tamper-proof.
3. Easy identification: each piece of data corresponds to a unique hash value and corresponds to a permanent object on the IPFS. As a result, as long as the reference to an object is maintained, access to the object is guaranteed.

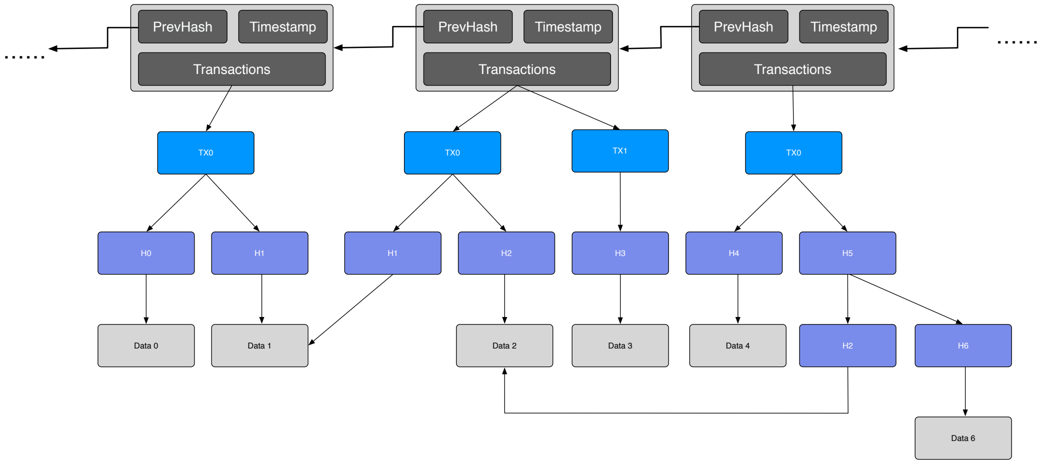

Figure 6: Referencing IPFS objects in Blockchain transactions.

4.2. Security and Privacy of Data Storage

Since IPFS by default is a global storage system, it is important that data should only be shared with only the limited and intended recipients. In order to achieve that, there are at least two options: the private IPFS network approach and the data encryption approach.

Contrary to the current most prevailing deployment of IPFS, *i.e.*, the `ipfs.io` network, it is possible to deploy IPFS in private network settings. Currently IPFS is under active development for accommodating private networks, and private IPFS networks can be achieved using a single shared symmetric encryption passphrase, then encrypt the transport layer with it. However, for maintaining the network, the service operator has to conduct key rotation procedures regularly in order to ensure long term security, especially when new odes join or leave the network, or service operators are updated.

Despite the feasibility of private IPFS network, it would not be the ideal solution to data security and privacy since such a setup heavily relies on how the private network is maintained. As a result, for better security, it is obligatory to implement a systematic data encryption approach.

Obviously, the reason why private IPFS network does not offer the maximum security options is because of the use of symmetric encryption, and the fact that management of the shared key can be a severe security vulnerability. In order to implement a secure data storage on top of the public and global IPFS storage, Ultiledger takes advantage of a range of asymmetric encryption algorithms, including public key technology, multisignature (Harn, 1994), group signature (Camenisch and Stadler, 1997) (as illustrated in Figure 7) and its successor ring signature scheme (Rivest et al., 2001).

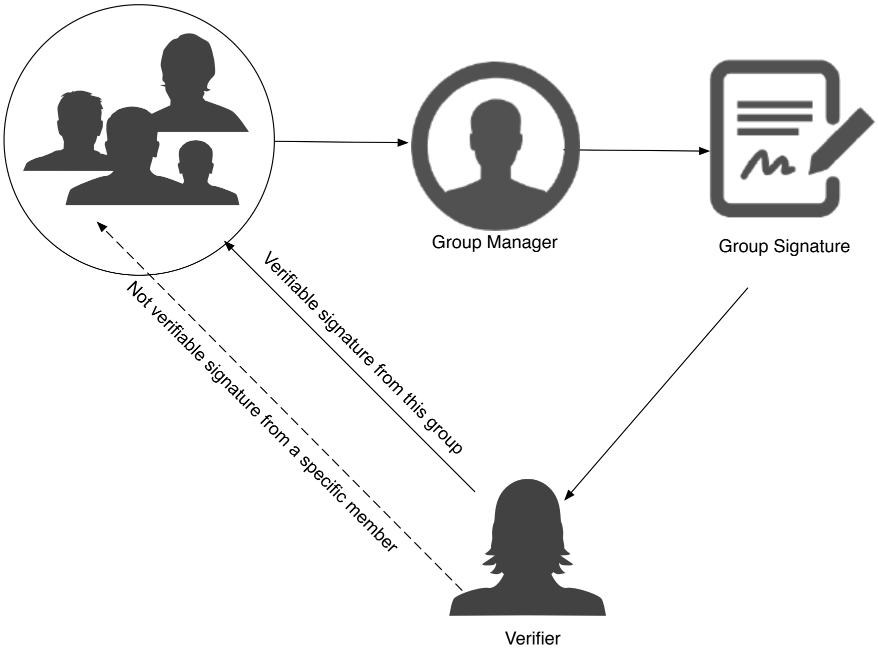
Group Manager
Group Signature
Verifiable signature from this group
Not verifiable signature from a specific member
Verifier

Figure 7: Illustration of the group signature algorithm.

More specifically, whenever a user sends a transaction to the Ultiledger blockchain or subsequently stores data onto the IPFS network, he has to encrypt the relevant data with the public keys of the recipients. Such a procedure prevents people other than the intended recipients accessing such data.

Normally, transactions on the Ultiledger network are single-signature transactions since they require only one signature from the owner of the data. However, in cases where the ownership of a piece of data is shared among a group of users, it becomes more complicated. In such cases, transactions require the signatures of multiple users and such procedures are tackled with multi signature algorithms, which ensures that only when all required signatures are collected is the data ready for IPFS.

Finally, when users are concerned with the identify privacy, *i.e.*, anonymity of transactions, they can opt in for group signature or ring signature algorithms. Simply put, these algorithms helps concealing the true identity of the sender of a transaction, by means of constructing an artificial identity based on the identities of a group of users.

To sum up, Ultiledger relies on the public global IPFS network as its underlying data storage platform. To ensure maximum data security and user privacy, a range of asymmetric encryption algorithms are adopted, so that data on IPFS can only be accessed by intended parties.

4.3. Full Nodes and Light Clients

In the Ultiledger network, it is essential that the blockchain is replicated across as many nodes as possible, in an as timely as possible manner so that the security of the transactions is ensured. However, in this network, it is practically not feasible to enforce every node to have the most advanced computation power and storage capability.

Especially, today mobile devices are becoming increasingly popular and neither the computing power nor the storage performance is comparable to enterprise hardware. As a result, when users bring in their own devices, it is mandatory that the Ultiledger blockchain can accommodate various types of hardware. To that end, we introduce another type of network nodes, namely light clients.

Contrary to full network nodes which replicates a full copy of the complete ledger and participate in the transaction validation task, light clients are network nodes in Ultiledger that have limited capacity and functionality. Essentially, light clients are capable of reading and accessing all data in the Ultiledger blockchain on the fly, with the help of a set of APIs. These APIs consist of the fundamental functionalities, including querying account balances, sending transactions, selecting the proxy full nodes, *etc.*.

By proxy of full nodes, light clients can still access a fairly large amount of services with the exception of transaction, which requires a node to have immediate access of the complete copy of the ledger as well as to have reliable network communication capabilities in order to synchronize the ledger with other nodes.

5. Smart Contracts

One part of Bitcoin that is often overlooked by users is not user-friendly scripting language. It is a specific and not Turing-complete programming language that contains nearly 200 commands (or `opcodes`), including cryptographic operations such as hashing and data validation. Since the scripting language is implemented without a standard specification (Bonneau et al., 2015) and it's hard to understand, very few people actually use Bitcoin's scripting language to implement smart contracts.

In order to provide users with a better smart contract development environment, Ethereum (Wood, 2014) implements a Turing-complete smart contract programming language named Solidity, and provides virtual machines to support development of smart contracts on the Ethereum blockchain. Although Ethereum has achieved great success, a majority of the smart contracts on Ethereum are intended for ICO purposes. Services from other categories, such as CryptoKitties[7], are not only rare, but also occasionally make the entire Ethereum network congested when a large number of users pour in.

Overall, the Ultiledger smart contract system possesses the following characteristics in comparison with existing smart contract systems:

- Virtual machine and custom `opcodes`
- Standard high-level libraries
- Upgradability

5.1. Virtual Machine and Custom Opcodes

Smart contracts are a specific type of computer program whose main functionality is to automate a predefined process according to a contract specification. Unlike normal computer programs that run on normal computing architectures, a smart contract is run on a blockchain infrastructure. That is, if we consider all the nodes in Ultiledger blockchain as a single computing entity, smart contracts runs on this exact entity.

Since the Ultiledger is not a standard computer, we have to design a virtual machine as an abstraction layer, so that each network node runs such a virtual machine. As a holistic entity, this virtual machine executes smart contracts deployed on Ultiledger blockchain.

In order to ensure the practicality of Ultiledger smart contracts, we propose a custom set of `opcodes` that are specific to the peer financing industry. Although Ethereum's vritual machine EVM[8] offers a large set of `opcodes`, most of them are generic arithmetic operations. Application specific operations are unfortunately missing. Ultiledger intends to fill this gap. For instance, Ultiledger virtual machine supports natively `transfer` as an atomic operation.

[7] https://www.cryptokitties.co/

[8] https://github.com/trailofbits/evm-opcodes

These opcodes that are customized according to specific while still generic applications will inevitably motivate developers to development more intelligent applications in an efficient manner. Furthermore, as a low-level function, such operations will be highly optimized, so that the execution speed is also guaranteed.

5.2. Standard Libraries

Contrary to , which are low level APIs executed by the virtual machine, in developers take advantage of a high-level Turing-complete language to implement smart contracts. After the developer is done with the implementation, contracts written in high-level programing language are compiled into virtual machine understandable and executed on the Ultiledger blockchain.

In order to empower developers with more productivity, Ultiledger provides a set of high level APIs on top of the `opcode set`. This is comparable to a paradigm shift with generic computer programing languages. For instance, the *C* programming language specification not only defines the reserved keywords, data types, arithmetic operands, lexical elements and grammar, but also a standard library including mathematics (`math.h`), signal handling (`signal.h`), input and output (`stdio.h`) as well as generic utilities (`stdlib.h`).

Lack of standard libraries, on the other hand, often lead to various issues and vulnerabilities. For instance, one notorious example is Parity's multisig library[9], which is a third-party library for accessing wallets and a great number of other smart contracts rely on its proper functioning. Unfortunately this library has a severe security vulnerability and has led to a loss of more than half a million ETH.

By providing a standard library, Ultiledger ensures that such a library is not only developer-friendly, but also risk-minimal in terms of security. Furthermore, such a standard library will definitely motivate developers implementing more applications based on smart contracts, thus help promoting the Ultiledger ecosystem.

5.3. Upgradability

Due to the tamper-proof and immutable nature of blockchains, existing smart contract systems generally do not support upgrading smart contract after it is deployed, even if a contract has significant security vulnerabilities. This situation has caused great inconvenience to developers because the contract update involves a series of data migration processes. Moreover, as the number of smart contract users increases, the difficulty and cost of contract renewal increases accordingly. To solve this type of problem, Ultiledger will support the update and fork function of smart contracts.

Specifically, since Ultiledger allows the user to store the smart contract to IPFS and only stores the address in the public account that points to the contract in IPFS, the contract update only needs to provide the address of the new version of the contract. If the user chooses to keep the old version of the contract because of non-security updates, Ultiledger still supports this possibility because the smart contract can only be updated and cannot be deleted. The process of updating a smart contract is very simple. The owner of the contract only needs to change the version number of the smart contract, upload the contract to IPFS and register the update in the public ledger. The Ultiledger system automatically monitors the compatibility of new and old versions of the contract and migrates user data.

Figure 8 illustrates an example where a smart contract is upgraded twice, from an initial `Version 0` upgraded to `Version 1` and then amended to `Version 2`. During the upgrade from `Version 0` to `Version 1`, the developer simply updates one module, while the unchanged parts remain intact and referenced to the corresponding objects. When upgrading from `Version 1` to `Version 2`, an extra module is incorporated, thus the original references remain intact and only a new reference is added to the smart contract DAG.

[9] https://etherscan.io/address/0x863df6bfa4469f3ead0be8f9f2aae51c91a907b4

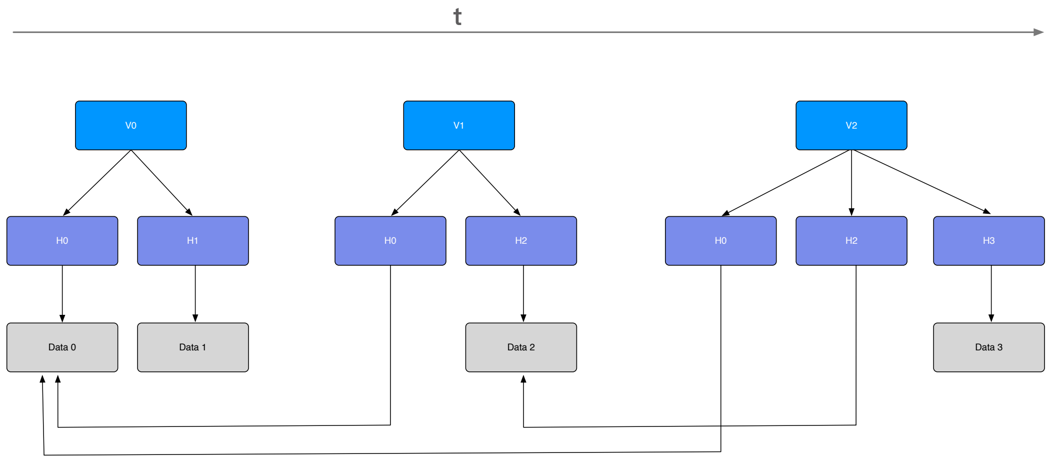

Figure 8: Smart contract upgrading over time.

6. Application Layer

In order to promote the Ultiledger ecosystem, we also incorporate a number of services in the application layer, namely:

1. Data storage abstraction layer: it provides users with a single standardized interface for storing and accessing their private data;
2. Authentication services: such services are used to provide convenient registration and authentication services for other modules in the ecosystem;
3. API: A set of application level interfaces provides developers with a common development foundation to facilitate the construction of third-party services.

Built on top of the underlying foundation of the Ultiledger blockchain, we offer application level services, *i.e.*, blockchain as a service to end users to provide best user experience in the blockchain domain.

6.1. Data Storage Abstraction

Data storage has always been a focus for many users and enterprises alike. Thanks to the seamless integration of the Ultiledger ecosystem, we are able to provide users with a data storage service on top of the blockchain infrastructure. Here users are able to store and access their private data easily and securely.

Furthermore, in order to boost the Ultiledger ecosystem, users can also opt in as a storage node within the Ultiledger network, where the user's node helps load balancing the underlying infrastructure in exchange of tokens. Thanks to the user-friendly abstraction and interfaces, Ultiledger can indeed serve as a decentralized service and ecosystem.

6.2. Authentication As A Service

Another important characteristic from the Ultiledger ecosystem is its compliance to regulatory requirements, as a consequence, we are able to authenticate users based on their digital signatures with public key technologies.

An authentication service is the key to enable a root of trust during setup of a blockchain node, especially in the hierarchical chain architecture where the internal transactions are not visible from other chains. As a result, it is fundamentally important to ensure the node enrollment and registration process with rigid authentication. In addition, the node operations such as node maintenance and management also require proper authentication and authorization.

In short, authentication service in Ultiledger will not only be secure, but also become a necessity within the ecosystem. Thus it is beneficial to provide authentication as a service to its end users.

6.3. API and SDK

It is straightforward that developers are an important and organic part of any technical systems, since they serve as both collaborators and a special type of end users. Moreover, good applications built on top of Ultiledger are likely to attract more users into the ecosystem. It is then important to build up a coherent technical community by providing a developer-friendly API and SDK.

Ultiledger's SDK will be fully open sourced, and developers are encouraged to contribute to improve the development experience by sharing code examples, tutorials, contribute to development documentation and submit pull requests. Developers with high contribution activities will be rewarded by Ultiledger with tokens and have the opportunity to join the team to work in full time.

7. Conclusion

This technical whitepaper has described the architecture of the Ultiledger blockchain, which aims to build a fast, low-cost, efficient, secure, and reliable blockchain economic ecosystem that meets large-scale daily business needs. The blockchain system brings innovation and improvements in three major layers: the consensus layer, the storage layer and the smart contract layer. Combined together, these three layers work in an complementary manner, so the Ultiledger can achieve high security, high scalability and TPS, and high application flexibility.

References

Androulaki, E., Barger, A., Bortnikov, V., Cachin, C., Christidis, K., De Caro, A., Enyeart, D., Ferris, C., Laventman, G., Manevich, Y., et al. (2018). Hyperledger fabric: a distributed operating system for permissioned blockchains. In *Proceedings of the Thirteenth EuroSys Conference*, page 30. ACM.

Avin, C. and Elsässer, R. (2013). Faster rumor spreading: Breaking the logn barrier. In *International Symposium on Distributed Computing*, pages 209–223. Springer.

Benet, J. (2014). Ipfs-content addressed, versioned, p2p file system. *arXiv preprint arXiv:1407.3561*.

Bonneau, J., Miller, A., Clark, J., Narayanan, A., Kroll, J. A., and Felten, E. W. (2015). Sok: Research perspectives and challenges for bitcoin and cryptocurrencies. In *Security and Privacy (SP), 2015 IEEE Symposium on*, pages 104–121. IEEE.

Boyd, S., Ghosh, A., Prabhakar, B., and Shah, D. (2006). Randomized gossip algorithms. *IEEE transactions on information theory*, 52(6):2508–2530.

Buterin, V. and Griffith, V. (2017). Casper the friendly finality gadget. *arXiv preprint arXiv:1710.09437*.

Camenisch, J. and Stadler, M. (1997). Efficient group signature schemes for large groups. In *Annual International Cryptology Conference*, pages 410–424. Springer.

Castro, M. and Liskov, B. (2002). Practical byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems (TOCS)*, 20(4):398–461.

Dias, D. and Benet, J. (2016). Distributed web applications with ipfs, tutorial. In *International Conference on Web Engineering*, pages 616–619. Springer.

Gilbert, S. and Lynch, N. (2002). Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *Acm Sigact News*, 33(2):51–59.

Harn, L. (1994). Group-oriented (t, n) threshold digital signature scheme and digital multisignature. *IEE Proceedings-Computers and Digital Techniques*, 141(5):307–313.

Karp, R., Schindelhauer, C., Shenker, S., and Vocking, B. (2000). Randomized rumor spreading. In *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, pages 565–574. IEEE.

Kiayias, A., Russell, A., David, B., and Oliynykov, R. (2017). Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Annual International Cryptology Conference*, pages 357–388. Springer.

King, S. and Nadal, S. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake.

Kwon, J. (2014). Tendermint: Consensus without mining. *Retrieved May*, 18:2017.

Larimer, D. (2014). Delegated proof-of-stake (dpos). *Bitshare whitepaper*.

Mazieres, D. (2015). The stellar consensus protocol: A federated model for internet-level consensus. *Stellar Development Foundation*.

Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system.

Nguyen, G.-T. and Kim, K. (2018). A survey about consensus algorithms used in blockchain. *Journal of Information Processing Systems*, 14(1).

Ongaro, D. and Ousterhout, J. K. (2014). In search of an understandable consensus algorithm. In *USENIX Annual Technical Conference*, pages 305–319.

Rivest, R. L., Shamir, A., and Tauman, Y. (2001). How to leak a secret. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 552–565. Springer.

Van Renesse, R., Dumitriu, D., Gough, V., and Thomas, C. (2008). Efficient reconciliation and flow control for anti-entropy protocols. In *proceedings of the 2nd Workshop on Large-Scale Distributed Systems and Middleware*, page 6. ACM.

Wood, G. (2014). Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151:1–32.