International Telecommunication Union

ITU-T Technical Specification

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

(1 AUG 2019)

ITU-T Focus Group on Application of
Distributed Ledger Technology
(FG DLT)

Technical Specification FG DLT D3.1

Distributed ledger technology reference architecture

FOREWORD

The International Telecommunication Union (ITU) is the United Nations specialized agency in the field of telecommunications, information and communication technologies (ICTs). The ITU Telecommunication Standardization Sector (ITU-T) is a permanent organ of ITU. ITU-T is responsible for studying technical, operating and tariff questions and issuing Recommendations on them with a view to standardizing telecommunications on a worldwide basis.

The procedures for establishment of focus groups are defined in Recommendation ITU-T A.7.

Deliverables of focus groups can take the form of technical reports, specifications, etc., and aim to provide material for consideration by the parent group in its standardization activities. Deliverables of focus groups are not ITU-T Recommendations.

The ITU Telecommunication Standardization Advisory Group established the ITU-T Focus Group on Application of Distributed Ledger Technology (FG DLT) in May 2017.

FG DLT concluded and adopted its Deliverables on 1 August 2019.

Type	**Number**	**Title**
Technical Specification	FG DLT D1.1	DLT terms and definitions
Technical Report	FG DLT D1.2	DLT overview, concepts, ecosystem
Technical Report	FG DLT D1.3	DLT standardization landscape
Technical Report	FG DLT D2.1	DLT use cases
Technical Specification	FG DLT D3.1	DLT reference architecture
Technical Specification	FG DLT D3.3	Assessment criteria for DLT platforms
Technical Report	FG DLT D4.1	DLT regulatory framework
Technical Report	FG DLT D5.1	Outlook on DLTs

The FG DLT Deliverables are available on the ITU webpage, at https://itu.int/en/ITU-T/focusgroups/dlt/.

For more information about FG DLT and its deliverables, please contact Martin Adolph (ITU) at tsbfgdlt@itu.int.

© ITU 2019

All rights reserved. No part of this publication may be reproduced, by any means whatsoever, without the prior written permission of ITU.

Technical Specification FG DLT D3.1

Distributed ledger technology reference architecture

Summary

This technical specification is a deliverable of the ITU-T Focus Group on Application of Distributed Ledger Technology (FG DLT).

It specifies the reference architecture for distributed ledger technology (DLT), the hierarchical relationship, specific functions and core components of the architecture. The applicability of the reference architecture is illustrated by mapping it to 14 live DLT platforms, including some of the most popular ones.

Acknowledgements

This Technical Specification was researched and principally authored by Ning Hu (Ontology), Wei Kai (CAICT), Ruifeng Hu (Huawei), Xiaofeng Chen (Qulian Technology). Giovanni Cambornero (ANCE), Skylar Hurwitz (Jelurida) and Prof. Javier Ibáñez (Comillas University) reviewed it.

Disclaimers

(1) Sample projects and reference articles mentioned in this deliverable are only for the purpose of analysis of technical architecture. The Focus Group does not endorse any of these projects and/or reference articles, nor their technical aspects.

(2) The editors have included these sample projects and/or reference articles based on the availability of mapping documents in attachments as examples for better explaining DLT. The inclusion of these examples does not imply any endorsement of, or judgement on, the quality or applicability of the mentioned implementations of the technology.

Keywords

DLT; distributed ledger technology; ledger; blockchain; reference architecture; architecture; components; functions; platforms

Editors:	Ning Hu Ontology China	Tel: E-mail:	+8618964015463 hehehu@gmail.com
	Wei Kai CAICT China	Tel: E-mail:	+8601062300249 weikai@caict.ac.cn
	Ruifeng Hu Huawei China	Tel: E-mail:	+8602556621725 huruifeng@huawei.com
	Xiaofeng Chen Qulian China	Tel: E-mail:	+8615088675364 chenxiaofeng@hyperchain.cn

CONTENTS

Technical Specification FG DLT D3.1

Distributed ledger technology reference architecture

1 Scope

This technical specification defines the reference architecture for distributed ledger technology (DLT), the hierarchical relationship and specific functions of the DLT architecture, important modules and specific functions in the structure of DLT, the main technical route and direction of the core module in the DLT.

It can be used as a guideline for DLT service providers to build system, and for the organizations to select and use a DLT platform.

2 Definitions

2.1 Terms defined elsewhere

This document uses the terms defined in [b-DLT 1.1].

2.2 Terms defined in this specification

This specification defines the following terms:

2.2.1 **event model**: Model where consensus takes place upon event (transaction).

2.2.2 **state model**: Model where consensus takes place upon state (result).

2.2.3 **UTXO model**: Model where transaction spends output from prior unspent transactions and generates new outputs that can be spent in the future. The unspent transactions are kept in each node.

2.2.4 **balance model**: Model that keeps track of the balance of each account as a global state. The balance of an account is checked to make sure it is larger than or equal to the spending transaction amount.

3 Abbreviations and acronyms

This specification uses the following abbreviations and acronyms:

AAA	Authentication, Authorization and Accounting
AES	Advanced Encryption Standard
API	Application Programming Interface
BFT	Byzantine Fault Tolerance Algorithm
CA	Certificate Authority
dBFT	Delegated Byzantine Fault Tolerance Algorithm
DApp	Decentralized Application
DES	Data Encryption Standard
DLT	Distributed Ledger Technology
DPoS	Delegated Proof of Stake
ECC	Elliptic Curves Cryptography
EVM	Ethereum Virtual Machine
IBE	Identity Based Encryption

JVM	Java Virtual Machine
KYC	Know Your Customer
P2P	Peer to Peer
PKI	Public Key Infrastructure
PoS	Proof of Stake
PoW	Proof of Work
RPC	Remote Procedure Call
SDK	Software Development Kit
SHA	Secure Hash Algorithm
SM3	Shang Mi 3[1]
TEE	Trust Execution Environment
TXO	Transaction Output
UTXO	Unspent Transaction Output
VBFT	Byzantine Fault Tolerance with Verifiable Randomness
VM	Virtual Machine

4 Conventions

This clause is intentionally left blank.

[1] One of the hash algorithms specified in [b-ISO/IEC 10118-3]

5 Architecture Overview

5.1 Overview

The high-level architecture constrains the highly abstract hierarchical architecture of distributed ledgers. The high-level architecture can cover almost all distributed ledgers, including public chains represented by Ethereum [b-ethe] and Bitcoin [b-bitc], private chains represented by Hyperledger Fabric and non-blockchain distributed ledgers systems.

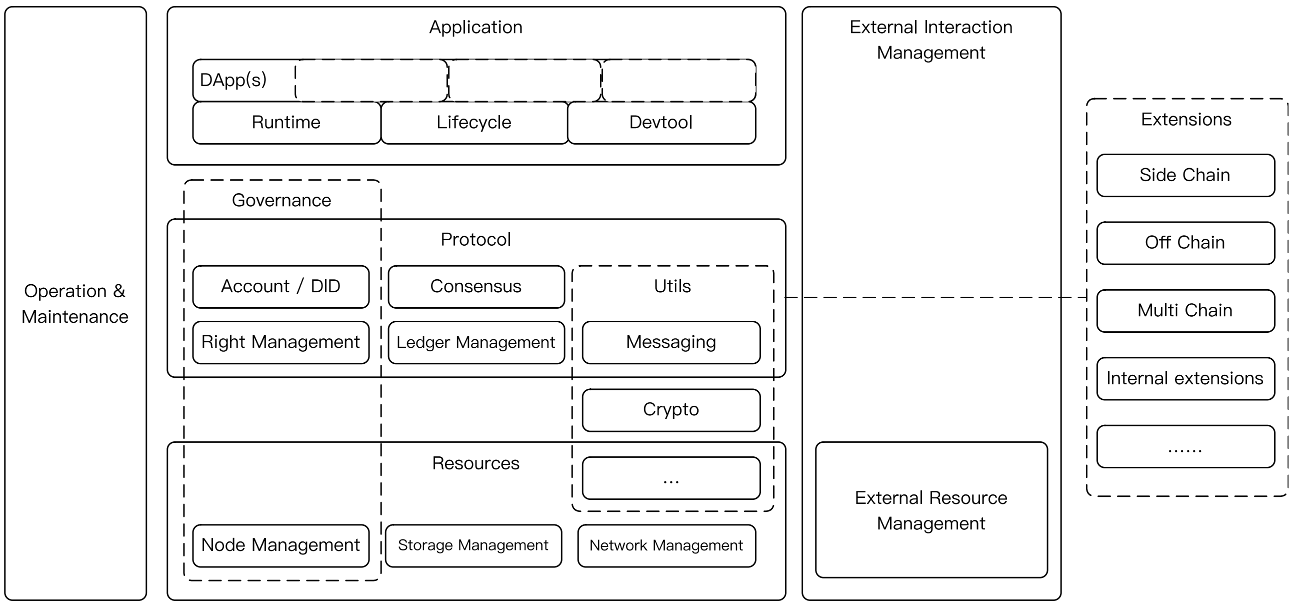

Figure 1: High-level conceptual architecture of DLT

5.2 Resource and infrastructure functions

The infrastructure provides the operating environment and basic components required for the normal operation of the distributed ledger system. The base layer includes network services, storage services, and computing services. The layer is the resource that most software systems rely on and is the foundational support of the distributed ledger system.

- **Network management functions -** Each DLT system is built upon a network hypothesis, which leads to the distribute model of the system. For example, in the study of bitcoin, each node inside bitcoin network has the same privileges, thus a P2P network model is used.
- **Storage management functions -** Each DLT system has a standard storage component to persist data and ensure data protection and privacy. In particular based on the cost of distributed storage, storage management may need to provide solutions for on-chain business to balance cost and data protection.
- **Utility functions -** Distributed ledger technology has utility functions to protect data – not only raw data, but also data transfer.
- **Node management functions -** Each node inside a DLT system is maintained by node owner/operators. Node management is a component to manage the resource of a single node inside a DLT system.

5.3 Protocol / governance and compliance functions

In DLT systems, blockchain systems in particular, each node can have its own implementations based on the system's technical specifications. The protocol layer is a conceptual layer to serve the technical specification across nodes inside a DLT system.

The protocol layer includes “governance (and compliance)”, “consensus”, “ledger management” and “messaging”. Furthermore, the “governance and compliance” includes “node management”, “AAA management” (“Account management” and “right management”). Its function is to support the management of system governance (based on trust endorsement hypothesis) and AAA functions of other components.

5.3.1 Consensus mechanism functions

A consensus mechanism is the core component of the distributed ledger, especially decentralized ledgers, and is used to ensure the consensus of all nodes on the data. The consensus mechanism contains data consistency algorithms (also known as consensus algorithms), data validation, data distribution and synchronization. By use of the consensus mechanism, the distributed ledger system sets up a trust mechanism upon the network hypothesis. Trust endorsement module, e.g., incentive mechanisms, is built upon that. Consensus mechanisms can be used to maintain public chain data and can also maintain data based on various distributed ledger partitioning mechanisms.

5.3.2 Ledger management functions

The ledger provides basic data management of distributed ledgers and distribution management of ledger data on the network. It defines the local data storage methods of ledgers and the synchronization mechanism between nodes and responds to rights management by use of consensus mechanism.

5.4 Decentralized application (DApp) functions

Based on the runtime management, DApps are built to serve different business requirements in a distributed network environment.

Furthermore, web applications with a combination of both off-chain services and on-chain services can be a solution for businesses.

5.4.1 Smart contract mechanism functions

DLT can support more complex transactions as technology evolves. Some complex transactions are stored in DLT systems in the form of source code/bytecode programs and can be executed to deal with different business logics. These programs are called smart contracts.

The smart contract mechanism includes language definition, compilation and execution of the code. Smart contracts for different DLT systems can be implemented using simple interpreted scripts or programming languages.

Smart contract mechanism is an optional but useful component for DLT.

5.4.2 DApp management functions

An open DApp management layer is a middleware, which connects the DLT network with the DApp business.

The DApp management middleware contains a DApp framework for DApp developers to create and maintain DApps, and a smart contract management mechanism for DApp hosts to manage their DApps easily.

Furthermore, the DApp framework can provide a series of interfaces for DApp developers. The interface provides the access to use of distributed ledgers. DApp users access the DLT services through the interface. The interface can usually have API, SDK, RPC, and so on.

5.5 Operation & maintenance functions

Operation and maintenance functions include various libraries such as log, monitoring, node/network management, and scaling libraries.

5.6 External interaction management functions

Each DLT system has its own network hypothesis, trust endorsement hypothesis and governance model. Thus, a DLT system with open-network hypothesis is able to interact/interoperate with external system(s).

In most cases, external interaction management is the runtime engine of external resource management.

5.7 Extension functions

The extension component of a DLT platform targets to resolve different requirements of data interoperability. “Extension functions” include a series of protocols/specifications for data interoperations of external systems, e.g., “multi-chain”, “side-chain”, “off-chain”, or internal systems, e.g., “child-chain”, “sharding” [b-shard1, b-shard2].

5.7.1 Internal system extensions

Each DLT system has one governance model. Internal system extension targets to resolve the problem of scalability for one DTL ecosystem, with the same governance model.

Child-chain

Child chains are individual ledgers with their own native tokens responsible for operational transactions such as deploying smart contracts, issuing assets, voting on polls and sending messages. All child chains receive consensus from, and share the same source code as, the network’s main (“parent”) chain; therefore, all child chains on the network are interoperable.

5.7.2 External system extensions

In most business cases, the data interoperability is a cross system requirement. A DLT system should be able to access external systems to satisfy business requirements.

Off-chain system(s) functions

In the study of DLT, to interact/interoperate with off-chain systems are scenarios in most use cases.

Furthermore, a series of L2 (layer 2) solutions, a combination of on-chain and off-chain techniques, are used to optimize the performance, as well as the scalability, of DLT systems.

6 Functional Components

The different distributed ledger platforms are highly consistent on the top-level architecture, but the components in the detailed architecture are different. The next part will explain in detail the components and functions of the detailed architecture, which varies slightly from the top-level architecture noted previously.

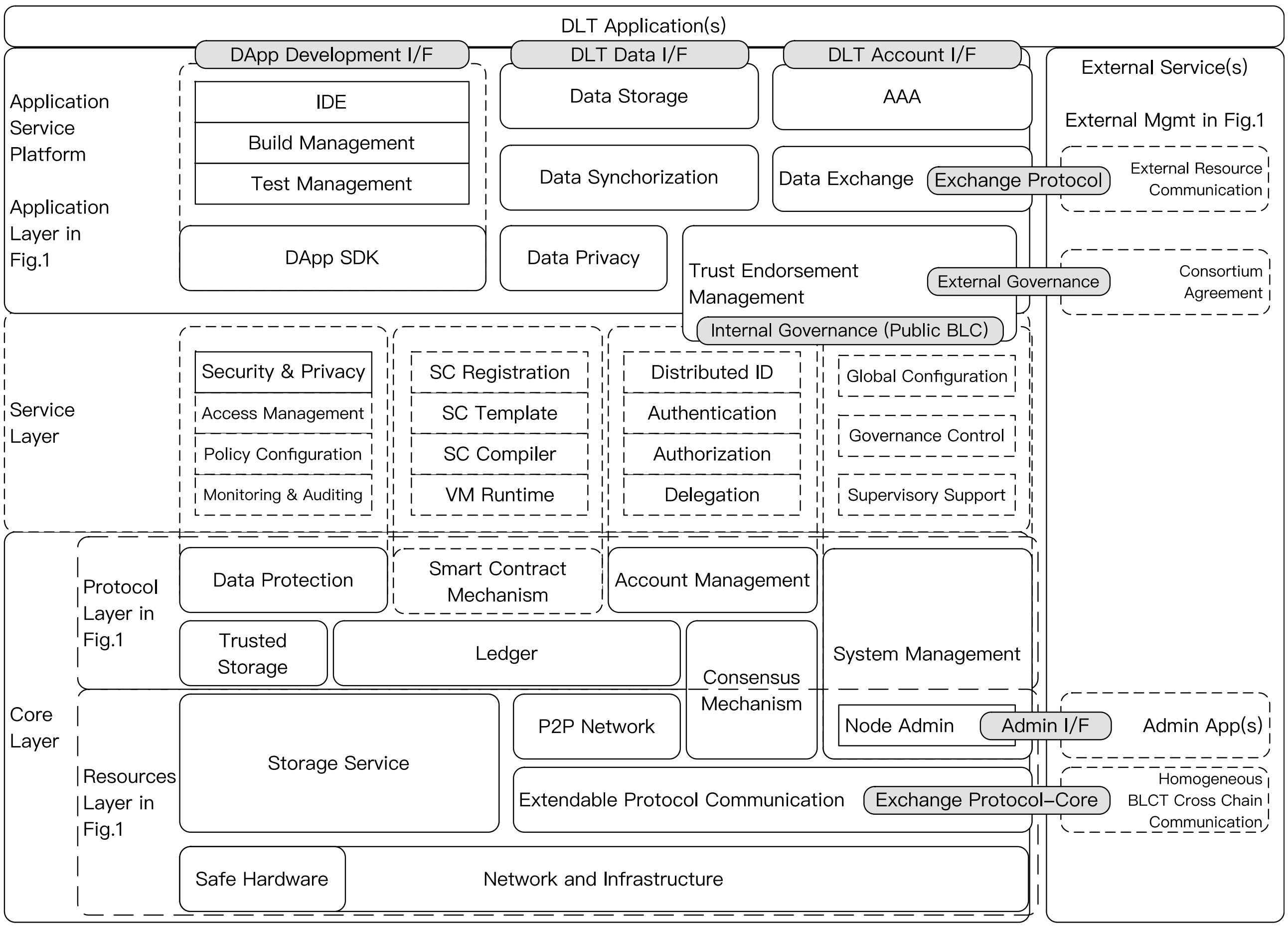

Figure 2: Architecture diagram

This part mainly explains the significance and role of each module of the detailed architecture.

6.1 Core layer

Functions map to “resources” and “Protocol”.

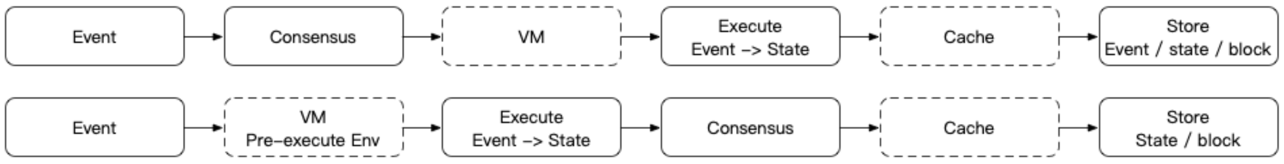

Figure 3: Typical flow of DLT systems

A typical DLT system targets to execute transactions (events) and store the result (state) in a distributed system.

The standard process involves:

- Event - to gather event from client(s);
- VM - to prepare the environment for event execution;

- Execute - to execute the transaction(s) inside the event, get state result. The state result stands for the status of ledger, supports different state models (UTXO, balance);
- Store - to store data (state, and/or event) into database(s).

In decentralized systems, a consensus mechanism is required to solve data consistency between different nodes.

There are two modes for approaching this in DLT systems:

- **State mode:** consensus upon states, mostly used for pre-execution, UTXO models.
- **Event mode:** consensus upon events, mostly used for post execution, balance models.

6.1.1 Network and infrastructure

As an IT solution, DLT nodes work with typical distributed system solutions (including cloud solutions).

6.1.1.1 Safe hardware

DLT systems work for data protection and data privacy with high performance where trusted/safe hardware can be optional, e.g., trust execution environment (TEE).

6.1.2 Extendable protocol communication

Scalable protocol communication module based on network hypothesis across distributed systems.

The communication component targets to give the capability of data exchange across different chain networks with L7 (layer 7) filtering for homogeneous DLT network if possible.

6.1.3 Network (P2P network) management

A DLT system is based on a network hypothesis, thus network management is required. Especially in permissionless DLT systems, each node has the same privileges, the node owner is able to decide the contribution of the node itself, network management is to provide basic capability to control node network.

6.1.3.1 Network discovery

In a distributed ledger system, there are usually many nodes, especially in public permissionless DLT systems. Each node needs to discover neighbor nodes through a network discovery protocol and establish a link with neighbor nodes. According to different distributed ledger architectures, the network discovery protocol also needs to identify and authenticate the node identity to prevent various attacks such as sybil attack.

6.1.3.2 Data transceiver

After the node connects to the neighbor node through the network discovery protocol, data exchange (e.g., transaction broadcast, consensus message, data synchronization) with other nodes is completed by the data transceiver module. According to the architecture of different distributed ledgers, the design of the data transceiver needs to consider requirements such as serial number, confirmation, and encryption.

6.1.4 Storage service

6.1.4.1 Data persistence

Distributed ledgers need to persistently store a variety of data, such as block data, transaction data, status data and private data of a local account. Depending on the type of data and the design of the distributed ledger, different storage modes can be used. Storage modes include relational databases such as MySQL [b-mysql], non-relational databases such as LevelDB [b-leveldb], and self-organizing files.

6.1.5 Consensus mechanism

6.1.5.1 Data synchronization

The data synchronization module ensures that the distributed ledger has a consistent ledger. The data synchronization module transmits the new part of the ledger between different nodes. The synchronization module also validates the synchronized data to ensure the correctness and consistency of the synchronized data.

Different types of nodes have different synchronization methods, which include synchronizing all transactions, blocks and status data, synchronizing all transactions and block data, synchronizing partial transaction data, and so on.

6.1.5.2 Legitimate validation

Consensus is the core of distributed ledgers. The purpose of consensus is to make many nodes involved in accounting jointly maintain a consistent account. It should be noted that the consensus algorithm here refers only to algorithms that agree on the transaction order, and not verification of the transaction data itself.

Consensus algorithm design should ensure the following:

- **Consistency:** Consensus nodes eventually need to agree on the data;
- **Timeliness:** Consensus nodes should complete the data consensus in as short a time as possible;
- **Security:** It takes a huge cost to undermine consistency and cannot be easily attacked.

In theory, all kinds of algorithms can meet the above requirements and be applied in a certain scene as the consensus algorithm of distributed ledger.

6.1.5.3 Consensus algorithm

PoW (proof of work) [b-pow], commonly known as mining, can be simply understood as providing a proof that you have done a certain amount of work. PoW requires the node to carry out a certain amount of computation to obtain the accounting right, which means that it takes a certain amount of energy to be consumed by the computer and calculates a verifiable result through the mathematical operation. The node sends the data that needs to be recorded in this round. After verification, the other nodes in the whole network store the data together.

The main feature of the PoW system is that the prover needs to do more work to get the result, while the verifier can easily verify whether the prover has done the corresponding work through the result. A core feature of this approach is asymmetry: work is more difficult for a prover and easier for approver (such as hash algorithm).

Series of Byzantine fault tolerance algorithm (BFT). Byzantine fault tolerance is a common solution to achieve efficient fault tolerance. The system comes from Byzantine general problem. The system based on Byzantine fault tolerance algorithm can reach consensus when the number of failure nodes or fraud nodes is less than the number of fault-tolerant nodes. Generally speaking, the number of fault-tolerant nodes is less than one-third of the total number of nodes.

At present, there are many implementations of BFT series algorithms. The typical implementation method is the state machine copy replica algorithm. Under the control of the master node, all nodes confirm the status of other participants through the three-phase protocol and determine the accounting according to the status of all the participants content. BFT series algorithms currently have synchronized PBFT algorithm [b-pbft], asynchronous Honey Badger BFT [b-hbft] algorithm, and open channel BFT algorithm, etc. There are also some BFT algorithms that support dynamic changes in the number of nodes.

PoS (proof of stake). PoS allows the so-called 'token holders' to replace the miners. The accountant is the holder of relevant tokens and accounting right is their 'Stake'. A typical PoS reduces the difficulty of mining in equal proportion to the percentage and time tokens that each node occupies in order to find the 'verifiable result' and determine the ownership of accounting rights faster.

DPoS (delegated proof of stake) [b-dpos]. Each token holder determines the accounting right of a DLT system by voting, similar to the election of the board of directors. All nodes whose votes exceed the agreed votes become system trustees, forming a "board of directors" and alternately signing blocks. There are incentives for encouraging directors never to miss signing a block, but if it occurs, the other nodes on the network may vote for a new director to take their place.

Traditional consensus algorithms. Traditional consensus algorithms are based on traditional distributed consensus techniques, including PAXOS [b-pxos], RAFT [b-raft], and others. There are many similarities between the traditional consensus algorithm and the other consensus algorithms, all of which are aimed at solving the data inconsistency caused by network failure, hardware failure and data loss in the system. The traditional consistency algorithm pays more attention to performance and has higher requirements on the environment. At present, the traditional consistency algorithm is also widely used in DLT systems.

Hybrid consensus algorithm: The current consensus algorithms have their strengths and weaknesses, so there are some cases where consensus algorithms are mixed. Including BFT-RAFT [b-brft], combined with the high performance of RAFT and support for fraud nodes; dBFT [b-dbft], authorized BFT algorithm, integrated BFT with election and authorization mechanism for more nodes scenes; VBFT [b-vbft], achieves chain scalability by consensus node selection with VRF, anti-attack ability by randomness and PoS, and fast state finality with BFT.

6.1.6 Smart contract mechanism

Smart contracts are programs written on the distributed ledger system, which encode the rules for specific types of distributed ledger system transactions in a way that can be validated and triggered by specific conditions.

The design of smart contracts usually includes the contract engine, contract code management and contract data management.

Contract code management is responsible for the operation of the deployment and storage of the contract code.

Contract data management maintains contract data and provides an access interface for the contract engine.

The contract engine can execute the contract code and maintain the contract data according to the contract code.

6.1.6.1 Verification

The module contains a transaction buffer pool for distributed ledgers and basic verification of transactions. Distributed ledgers typically verify the sender's balance of the transaction, the sender's transaction number, and so on. Once the transaction is validated, the transaction is saved in the DLT system. The verification module can effectively prevent various attacks and ensure the stable operation of the distributed ledger.

6.1.6.2 Language and compiler

For very limited business requirements, you can use a script to implement a contract, which is fast, secure and low in resource consumption.

For lightweight businesses, an inline contract engine is recommended which is highly efficient and reduces the complexity of deployment.

For heavyweight businesses, an external execution environment is recommended which has more resources and higher execution efficiency. These are generally compatible with mainstream programming languages and are easy to develop.

Language and compiler provide the grammar specification of the contract as well as the compilation specification to ensure that the transaction can be processed by the execution engine.

6.1.6.3 Execution engine

External contract execution environment: It uses the external execution environment to execute the contract code, and usually uses an external container (such as Docker [b-dock]) or virtual machine to ensure a consistent execution environment for all nodes. The maintenance of contract data is accomplished by the distributed ledger process communicating with the agent process in the external environment, typical representative includes the realization of the contract in the Docker container of Hyperledger.

The advantages of this contractual technology are that they are Turing complete, utilize easy-to-master programming languages and offer efficient contract development. The heavyweight business is more efficient due to the permanent contractual execution environment. The disadvantages are slow deployment of contract code, weak external engine attack resistance, large resource usage, large inter-process communication overhead, and not suitable for short contract codes.

Inline contract engine: A contract engine is embedded in the process, and the contract codes are interpreted during execution or implemented after being compiled. The engine can use open third-party engines such as JVM [b-jvm], LUA [b-lua], JS [b-js] and other scripts; or be implemented by yourself, such as EVM [b-evm]. At present, most distributed ledgers use this technology, e.g., Ethereum.

The advantages of inline contract engines include that they are Turing complete, offer fast contract deployment, speed up data access, harden security of the engine, and provide convenient management and customization of the computing resources of the contract engine. The disadvantages are that the contract is generally not resident in memory, and the engine efficiency is difficult to compare with the external engine, making it less efficient to perform heavyweight business.

Scripting/Finite State Machine: A set of execution instructions is preset in the distributed ledger, this sequence forms a piece of contract code, which is interpreted by the state machine. Due to the complexity of the instructions and the size of the code, the script generally has a weak function and is not Turing complete, meaning it only performs basic operations. Many current virtual currency ledgers, including bitcoin, use this technology.

Advantages include that scripting technology is easy to implement, its script functions are effectively controlled, and there is strong anti-attack mitigation. Disadvantages include complex script development, non-Turing complete, and the scale of scripts is small, resulting in limited functions.

6.1.7 Ledger management

The ledger stores all the transaction data and contract data. The ledger needs to be able to complete the transaction and the processing, indexing, and storage of contracts.

Ledger management contains two components:

- Ledger mechanism - based on consensus mechanism, relying on P2P networking and extended storage services;
- Trusted storage - extended storage services for ledger.

Consensus algorithm design should ensure the following:

- There are complete definitions of the various data types in the ledger, such as transactions, blockchain, assets, accounts, etc.;
- The transaction data in each block of the ledger can be summed up using algorithms, such as Merkle tree, the blocks can be continuously added, and successive blocks are chain-linked to ensure that they cannot be tampered with;
- The design of the ledger can be used for data synchronization and verification and the ledgers should support large-scale data storage and high concurrency.

6.1.7.1 Transaction record

Account: The account must use a public-private key generated through an asymmetric encryption mechanism.

Assets: Assets can be expressed in the form of account and balance models, or in the form of TXO (e.g., UTXO [b-utxo]) model, or a combination of the two. Assets can also be defined in the contract data independent of the underlying assets on the ledger.

Ledger storage: The ledger storage is usually implemented as a custom file or as an embedded database. The ledger store is divided into two parts, one part is the block data, which contains the original transaction, the other part is the result of the transaction or contract execution, usually stored outside the block.

Block structure: block consists of the transactions, a block header, a variety of relevant data, and its own block digest. The digest of the block is usually calculated from the transactions in the block, the digest of the previous block, and various other related data according to an algorithm such as the Merkle tree.

6.1.7.2 Status

Most distributed ledgers, in addition to saving blocks and transaction data, also hold some data or results of the transaction execution, which can be called Status. Because of the consistency of the distributed ledger smart contract engine execution, the status of different nodes of the distributed ledger can be consistent. For a simple distributed ledger like Bitcoin, the status may contain only a list of unspent assets, and so on. For distributed ledgers like Ethereum and Hyperledger with Turing-complete intelligent contract engines, the status preserves more complex data generated during virtual machine execution.

By maintaining status data, distributed ledgers can continue to process a series of complex and continuous transactions.

6.1.8 Data protection

DLT platforms should provide data protection capabilities for data reading from and writing to ledgers. They should use trusted storage to ensure that data is tamperproof.

6.1.9 Utility

6.1.9.1 Cipher library

Distributed ledgers use a variety of cryptographic algorithms. The cryptographic algorithm library provides basic cryptographic algorithm support for each component, including various commonly used encoding algorithms, hash algorithms, signature algorithms, privacy protection algorithms, etc. The cryptographic algorithm library also provides functions such as maintenance and storage of secret keys.

6.1.9.2 Messaging

The message module provides message notification services between different components within the distributed ledger and between different nodes. For example, after a successful transaction, the customer usually needs to track the results of the execution of the transaction and even some records during the execution of the transaction. The message module can complete the generation, distribution, storage and other functions of the message to meet the needs of the distributed ledger.

6.2 Service layer

Functions map to "protocol" and "governance".

The core layer provides four basic services for upper applications, data protection, data processing management, data AAA management and infrastructure management. These four services constitute the middleware between DLT core and DApps.

6.2.1 Account management

Distributed account system serves all kinds of entities and data in a DLT system. The account management component controls addresses and identities.

- Address is bound with trust data. Especially in public chain projects, most trust data are usually digital asset.
- Identity is bound with entity.

Based on distributed identity and multi-dimensional authentication protocol, the account management component provides the identity management of node operators, trust endorsement regulatory roles, DApp operators, end users and extended Internet of Things access entities.

The account management of node identity usually utilizes the following technologies.

- **CA**: the certificates are issued through centralized CA for various applications in the system, and both the identity and authority management are certified and confirmed by these certificates;
- **IBE**: the identities are confirmed by IBE;
- **PKI**: the identities are confirmed by addresses/accounts based on PKI;
- **Third party identity authentication**: the identities are confirmed by the third party.

6.2.1.1 Distributed identity

Distributed identity establishes a cryptographic-based digital identity for business entities in a DLT system. The digital identity is based on DLT and is not subject to any centralized organization. It is potentially controlled by multiple entities and is both secure and trustworthy.

6.2.1.2 Authentication and authorization

AAA (accounting, authentication and authorization) solutions support the full use of account management.

Approaches to authority management differ since a variety of distributed ledgers have different application scenarios.

The existing business expansions can interface with the existing authentication and authority management.

The more concentrated authority management business can be complemented by using CA/IBE.

For instance, the public chains usually adapt PKI technology rather than the central authority management like CA/IBE.

6.2.1.3 Delegation

In DLT account management, non-user entities are also using distributed identities. The use of non-user entities shall enable ID delegation, where the entity owners can make full use of them.

6.2.2 Node (system) management

System management includes secure communication, trusted data transmission, and related services of node operation and network governance.

6.2.2.1 System configuration

System configuration for nodes inside a DLT system includes network configuration, communication management configuration, global system configuration, etc.

6.2.2.2 Node management

Node management for nodes inside DLT systems.

6.2.2.3 Governance control

Governance control for node, and governance control to a DLT system via multiple nodes, including network status, communication channel monitoring, alarm and tracking, trust endorsement, node failure monitoring, etc.

6.2.2.4 Supervisory support

Component support for the monitoring and supervision of dishonest events in a DLT system.

6.2.3 Smart contract mechanism

The smart contract mechanism handles trusted data processing, including smart contract lifecycle management, contract registration, pre-authorization, deployment, upgrading, iteration, and cancellation. In the service layer, the smart contract mechanism provides the component for contract registration, contract template, contract compiler and VM runtime.

6.2.4 Data protection

6.2.4.1 Policy configuration

The policy configuration of trusted data includes access templates, privacy templates, monitoring and auditing strategies, etc.

6.2.4.2 Access control

Trusted data access control and dynamic data operation control.

6.2.4.3 Data security and privacy management

Security management of data storage and privacy management for data affirmative easement.

6.2.4.4 Data monitoring and auditing

Tracking and monitoring of data, data processing, review and audit of special data events.

6.2.5 Trust endorsement management

According to the different trust endorsement approaches of distributed systems, trust endorsement management is used to meet the governance model. E.g., the legal endorsement of off-chain governance (consortium chains); the tokenomics endorsement of in-chain governance (public chains).

6.2.5.1 Incentive module

The incentive module is used for accounting incentives for partially distributed ledgers such as typical public chains. Most of the public chains use consensus algorithms for targeted token distribution and motivating the accounting nodes to account. In general, the incentive module and the consistency module are very closely related.

6.3 Application service platform

Functions map to "application" and "operation and maintenance".

6.3.1 DApp framework

DApp framework includes interfaces which support DApp development, DLT data management and DLT account management.

The upper layer interface provided by the distributed ledger gives external systems efficient access to the distributed ledger data, to external applications to integrate distributed ledgers or to other distributed ledgers for mutual access, usually including RPC, API and SDK. The interface layer mainly completes data synchronization, transaction exchange, etc.

The RPC interface connects peripheral components with the distributed ledger nodes over the network and to access the services provided by the distributed ledger. SDK provides a development package for other components to integrate part of the functionality of a distributed ledger.

RPC and SDK should observe the following rules:

- **Completely functional**: the transactions of distributed ledger can be completed and maintained, and an intervention strategy and privilege management are operational;
- **Portable**: it can be used in a variety of applications and environment, and is not limited to one absolute software or hardware platform;
- **Extensible and compatible**: it should be as forward and backward compatible as possible, and try not to modify or minimize changes as extending functions;
- **Easy to use**: the structured design and good naming methods should be used to reduce the cost of development.

Common implementation techniques include call control, serialized objects, and network components. There are various architectures which can be used, such as CORBA [b-corb], JsonRPC [b-jrpc], gRPC [b-grpc], Thrift [b-thrft], RestAPI [b-rapi], XMLRPC [b-xrpc] etc.

6.3.2 Accounting, authorization and authentication

The AAA system manages the DApp users ability to access data, process data and perform data exchange based on transactions.

The AAA management shall focus on three parts:

- Access control for DApp users to submit their transaction;
- Access control for nodes to access the chain network (permission control), usually being applied by private chains and consortium chain (permissioned chain);
- Privilege control for DApp users to operate the data and function calls via transactions.

6.3.3 Data privacy

The use of cryptographic techniques to protect on-chain user data and to meet data privacy requirement for applications (result in DApps).

Privacy has always been one of the obstacles to the application of distributed ledger. How to satisfy regulatory requirements and not infringe data privacy is the key to the distributed ledger industry.

Privacy protection should therefore meet the following requirements:

- **Anonymity controls**: to ensure that users can set the transaction so it is not visible to unrelated parties;
- **High-performance**: privacy-protected design must still be able to meet performance requirements;
- **Transparent supervision**: privacy protection should not evade the regulatory functions of regulatory agencies.

6.3.4 Data storage and synchronization

Useful tools for end users and DApps to facilitate DApp data storage, synchronization, operation and credentials.

6.3.5 Operation and maintenance

6.3.5.1 Deployment

The deployment of distributed ledger refers to the installation and use of distributed ledger services for different scenarios and users with different node permissions and service modes. According to the type, distributed ledgers are divided into public chains, consortium chains, and private chains, where their deployment methods are not the same.

Permissionless DLT

Permissionless DLTs generally do not make any restrictions to nodes access, and less demanding on the operating environment, the ledger nodes are relatively simple, all nodes are free to participate in consensus and read and write data.

Permissioned DLT

In such distributed ledgers, consensus processes can only be involved with authorized customer nodes. Authorized nodes can participate in the consensus and data read and write process according to the rules. Permissioned DLTs generally need to provide higher performance, so the operating environment requirements for consensus nodes are higher. It is recommended that such distributed ledgers use a high performance and consistent execution environment for deployment for higher performance and reliability.

The deployment of permissioned DLTs can occur in the following ways:

- **Process-based deployment -** Operations staff personnel deploy nodes on the host or virtual machine to complete the configuration. This deployment is flexible. However, due to the complexity of the configuration file, the deployment is inefficient and prone to problems caused by inconsistent node environments.
- **Container-based deployment -** This involves ensuring a uniform environment within the container and encapsulating the distributed ledger in the container, then deploying the container by the operations staff. This approach is simpler to deploy than process-based deployment and more reliable.
- **Cloud service-based deployment -** Integration of distributed ledger and cloud services that facilitates rapid deployment of distributed ledger services through the cloud platform, while providing different levels of PaaS, BaaS services.

6.4 DLT applications

Functions map to "application".

Multiple applications based on DLT systems, especially blockchain systems, DApps.

6.5 External services

Functions map to “external interaction management” and “extensions”.

The reference architecture of DLT, especially blockchain system, shall meet the requirement to balance the needs of security, decentralization, and scalability. Furthermore, decentralized systems are focusing on resolving “trust” issues in competitive business environments, therefore, not all business cases shall use decentralized systems.

A hybrid system combines DLT systems with typical IT systems and can satisfy most business requirements.

From a DLT perspective, external services provide solutions to cooperate with external systems.

6.5.1 Extensions

Extensions include the capability to interact with non-DLT systems, 3rd party DLT systems and/or layer 2 blockchain technology.

Concerning layer 2 blockchain technology, its main purpose is to scale blockchain transaction capacity while retaining the benefits decentralization brings to a distributed protocol. Solving the scalability problem will significantly help with blockchain’s mainstream adoption. Layer 2 blockchain technology systems are those that connect to and rely on blockchain systems as a base layer of security and finality.

Layer 2 solutions include plasma, state channel, sharding, raiden network, lightening network, etc.

6.5.2 External interaction/interoperation management

In layer 2 solutions, the blockchain system is able to interact/interoperate with layer 2 systems.

6.5.3 External resource management

To cooperate with non-DLT systems and third party DLT systems, mostly data/resource exchange transactions, resource management is required.

7 Architecture mapping of other distributed ledgers

See Annex A and electronic attachments.

Annex A: Overview of architecture mapping to existing DLT platforms

The applicability of the DLT reference architecture is illustrated by mapping it to 14 live DLT platforms, including some of the most popular ones.

Table A.1 provides an overview of the mapping. The electronic attachment to this technical specification contains the individual platform mappings.

Table A.1: Overview of architecture mapping to existing DLT platforms

Attachment	**Platform**	**Contributor**	**Organization**	**Reviewer**	**Organization**
I	**Alastria (Quorum version)**	Jesus Ruiz	Alastria	Paulo Brizola	Multiledgers
II	**Ardor**	Skylar Hurwitz	Jelurida	Xiaofeng Chen	Qulian
III	**Bitcoin**	Robin Renwick	Independent	Lisa Tan	Economics Design
IV	**Corda**	Paulo Brizola	Multiledgers	Ruifeng Hu	Huawei
V	**EOS**	Ning Hu	Ontology	Giovanni Cambronero	ANCE
VI	**Ethereum**	Suzana Maranhão	BNDES	Ning Hu	Ontology
VII	**Fabric**	Ruifeng Hu	Huawei	Paulo Brizola	Multiledgers
VIII	**Hyperchain**	Xiaofeng Chen	Qulian	Baixue Yang	CAICT
IX	**LACChain**	Marcos Allende	IADB	Ismael Arribas	Kunfud
X	**Masterchain**	Alexander Chuburkov	Russian Fintech Association	Lisa Tan	Economics Design
XI	**Monero**	Robin Renwick	Independent	Lisa Tan	Economics Design
XII	**Ontology**	Ning Hu	Ontology	Baixue Yang	CAICT
XIII	**Quorum**	Ismael Arribas / Jose Nogueira	Kunfud/BNDES	Paulo Brizola	Multiledgers
XIV	**Sawtooth**	Ruifeng Hu	Huawei	Xiaofeng Chen	Qulian

Bibliography

[b-bitc]	Bitcoin [online]. Available at: https://bitcoin.org/en/.
[b-brft]	Clow, J. & Jiang, Z. (2017). *A Byzantine Fault Tolerant Raft.* Stanford University. Available at: http://www.scs.stanford.edu/17au-cs244b/labs/projects/clow_jiang.pdf.
[b-corb]	CORBA [online]. Available at : http://www.corba.org/.
[b-dbft]	NEO [online]. Available at: https://docs.neo.org/docs/en-us/basic/technology/dbft.html .
[b-DLT 1.1]	ITU-T Technical Specification FG DLT D1.1 (2019), *DLT terms and definitions.*
[b-dock]	Docker [online]. Available at: https://www.docker.com/.
[b-dpos]	Bitshares (2019). *Delegated Proof-of-Stake Consensus*. Available at: https://bitshares.org/technology/delegated-proof-of-stake-consensus/.
[b-ethe]	Ethereum [online]. Available at: https://www.ethereum.org/.
[b-evm]	Ethereum (2018). *Awesome Ethereum Virtual Machine*. Available at: https://github.com/ethereum/wiki/wiki/Ethereum-Virtual-Machine-(EVM)-Awesome-List.
[b-grpc]	gRPC [online]. Available at: https://grpc.io/.
[b-hbft]	Miller, A. (2018) *The Honey Badger of BFT Protocols*. Available at: https://github.com/amiller/HoneyBadgerBFT.
[b-ISO/IEC 10118-3]	ISO/IEC 10118-3:2018, *IT Security techniques -- Hash-functions -- Part 3: Dedicated hash-functions.*
[b-jrpc]	JSON-RPC [online]. Available at: https://www.jsonrpc.org/.
[b-js]	JavaScript [online]. Available at: https://www.javascript.com/.
[b-jvm]	Tyson, M. (2018). *What is the JVM? Introducing the Java Virtual Machine.* The Java Platform Series. Available at: https://www.javaworld.com/article/3272244/core-java/what-is-the-jvm-introducing-the-java-virtual-machine.html.
[b-leveldb]	Ghemawat, S. & Dean, J. (2019). *LevelDB.* Available at: http://leveldb.org/.
[b-lua]	Lua [online]. Available at: https://www.lua.org/.
[b-mysql]	MySQL [online]. Available at: https://www.mysql.com/.
[b-pxos]	Lamport, L. (1998). *The Part-Time Parliament.* ACM Transactions on Computer Systems, Vol. 16, No. 2, pp. 133-169. Available at: http://lamport.azurewebsites.net/pubs/lamport-paxos.pdf.
[b-raft]	Raft (2013). *The Raft Consensus Algorithm*. Available at: https://raft.github.io/.
[b-rapi]	RESTfulAPI.net (2017). *What is REST* . Available at : https://restfulapi.net/.

[b-shard1]	Ethereum (2019). *On Sharding Blockchains* Available at: https://github.com/ethereum/wiki/wiki/Sharding-FAQs.
[b-shard2]	Ontology (2018). *Ontology Sharding Draft v0.2.* Available at: https://github.com/ontio/documentation/blob/master/sharding/ontology-sharding.pdf.
[b-thrft]	Apache Software Foundation (2017). *Apache Thrift*. Available at: http://thrift.apache.org/.
[b-utxo]	Bitcoin (2019). *Unspent Transaction Output, UTXO*. Available at: https://bitcoin.org/en/glossary/unspent-transaction-output.
[b-vbft]	Ontology (2019). *VBFT Introduction*. Available at: https://ontio.github.io/documentation/vbft_intro_en.html.
[b-xrpc]	XML-RPC [online]. Available at: http://xmlrpc.scripting.com/.

ITU-T