23

Alternative Blockchains

Introduction

This is an introduction to alternative blockchain solutions. With the success of Bitcoin and the subsequent realization of the potential of blockchain technology, something of a Cambrian explosion happened in the tech world, which resulted in the development of various blockchain protocols, applications, and platforms. Some projects did not gain much traction, but many have succeeded in securing a stable place in this space.

We will explore:

- Kadena
- EOS
- Tezos
- Solana
- Ripple
- Stellar
- MaidSafe
- Other platforms

Let's start with Kadena.

Kadena

Kadena is a blockchain that has successfully addressed scalability and privacy issues in blockchain systems. A new Turing-incomplete language, called **Pact**, has also been introduced with Kadena that allows the development of smart contracts. A key innovation in Kadena is its scalable BFT consensus algorithm, which has the potential to scale to thousands of nodes without performance degradation.

Scalable BFT is based on the original Raft algorithm and is a successor of Tangaroa and Juno. Tangaroa, which is the name given to an implementation of **Raft** with BFT, was developed to address the availability and safety issues that arose from the behavior of **Byzantine nodes** in the Raft algorithm, and Juno was a fork of Tangaroa. Both Tangaroa and Juno have a fundamental limitation—they cannot scale. As such, Juno and Tangaroa could not gain much traction. Tangaroa could scale up to 50 nodes with a processing speed of around 5,000 transactions per second, but it was only a **Proof of Concept** (**PoC**) and later it was discovered that it had some safety and liveness issues. Blockchains have the more desirable property of maintaining high performance as the number of nodes increases, but the aforementioned proposals lack this feature. Kadena solves these issues with its proprietary scalable BFT algorithm, which, according to the official documentation on Kadena, scales up to thousands of nodes without any performance degradation.

Moreover, confidentiality is another significant aspect of Kadena, and it enables the privacy of transactions on the blockchain. This security service is achieved by using a combination of key rotation, symmetric on-chain encryption, incremental hashing, and the **Double Ratchet** protocol:

- Key rotation is used as a standard mechanism to ensure the security of the private blockchain. It is used as a best practice to thwart any attacks if the keys have been compromised by periodically changing the encryption keys. There is native support for key rotation in the Pact smart contract language.
- Symmetric on-chain encryption allows the encryption of transaction data on the blockchain. These transactions can be automatically decrypted by the participants of a particular private transaction.
- Incremental hash functions are useful in situations where, if a message that was previously hashed changes slightly into a new message, then instead of recomputing a new hash from scratch again, the hash generated originally for the message is used to generate a new hash. This method is faster than generating a new hash altogether for the new message.
- The Double Ratchet protocol is used to provide key management and encryption functions.

The scalable BFT consensus protocol ensures that adequate replication and consensus have been achieved before smart contract execution. The consensus is achieved with the following process, which is how a transaction originates and flows in the network:

1. First, a new transaction is signed by the user and broadcast over the blockchain network, which is picked up by a leader node that adds it to its immutable log. At this point, an incremental hash is also calculated for the log. An incremental hash is a type of hash function that allows the computation of hash messages in the scenario where if a previous original message that is already hashed is slightly changed, then the new hash message is computed from the already existing hash. This scheme is quicker and less resource-intensive compared to a conventional hash function, where an altogether new hash message is required to be generated even if the original message has only changed very slightly.
2. Once the transaction is written to the log by the leader node, it signs the replication and incremental hash and broadcasts it to other nodes.

3. Other nodes, after receiving the transaction, verify the signature of the leader node, add the transaction into their own logs, and broadcast their own calculated incremental hashes (quorum proofs) to other nodes. Finally, the transaction is committed to the ledger permanently after an adequate number of proofs are received from other nodes. As the scalable BFT is deterministic, the transactions are not rolled back once committed. Each node will commit only if it sees that a majority of clusters are in agreement and evidence of the agreement is available. Otherwise, if this evidence of agreement is missing, the nodes will not do anything.

A simplified version of this process is shown in the following diagram, where the leader node is recording the new transactions and then replicating them to the follower nodes:

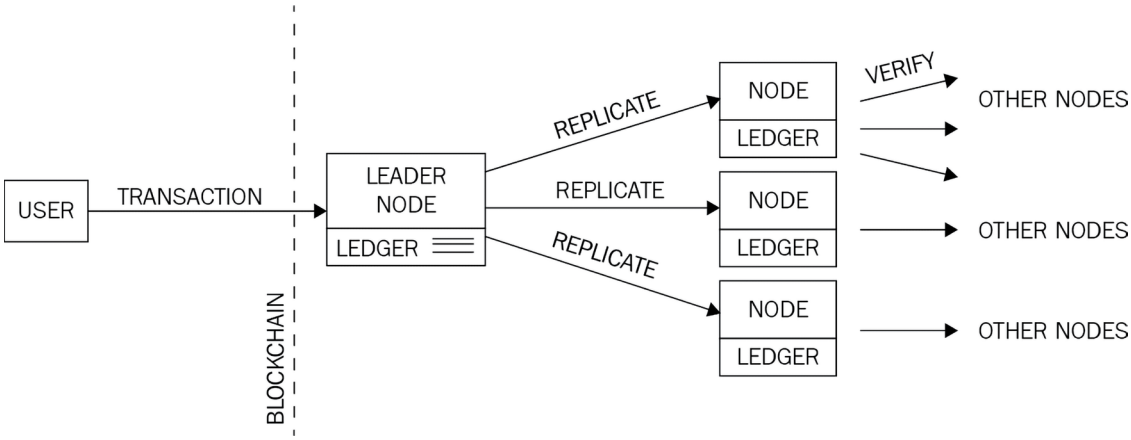

Figure 22.1: Consensus mechanism in Kadena

Once consensus is achieved, a smart contract execution can start and takes a number of steps, as follows:

1. The signature of the message is verified.
2. The Pact smart contract layer takes over.
3. The Pact code is compiled.
4. The transaction is initiated and executes any business logic embedded within the smart contract. In case of any failures, an immediate rollback is initiated that reverts that state back to what it was before the execution started.
5. Finally, the transaction completes, and the relevant logs are updated.

Pact has been open sourced by Kadena and is available for download at `http://kadena.io/pact/downloads.html`.

Pact

Pact can be downloaded as a standalone binary that provides a REPL for the language. An example is shown here, where Pact is run by issuing the `pact` command in the Linux console:

```
drequinox@drequinox-OP7010:~/Downloads$ . /pact
pact> 1234
1234
pact> (+ 1 2)
3
pact> (if (= (± 1 2) 3 "OK" "ERROR")
(interactive) :1:31: error: unexpected
    EOF, expected: ")", ";", "{",
    Boolean false, Boolean true,
    Decimal literal, Integer literal,
    String literal, Symbol literal,
    list literal, pact, sexp, space
(if (= (+ 1 2) 3 "OK" "ERROR")<E0F>

pact> (if (= (+1 2) 3) "OK" "ERROR")
"OK"
pact>
```

Figure 22.2: Pact REPL, showing sample commands and error output

A smart contract in the Pact language is usually composed of three sections: keysets, modules, and tables. These sections are described here:

- **Keysets**: This section defines relevant authorization schemes for modules and tables.
- **Modules**: This section defines the smart contract code encompassing the business logic in the form of functions and pacts. Pacts within modules are composed of multiple steps and are executed sequentially.
- **Tables**: This section is an access-controlled construct defined within modules. Only administrators defined in the admin keyset have direct access to this table. Code within the module is granted full access, by default, to the tables.

Pact also allows several execution modes. These modes include contract definition, transaction execution, and querying. These execution modes are described here:

- **Contract definition**: This mode allows a contract to be created on the blockchain via a single transaction message.
- **Transaction execution**: This mode entails the execution of modules of smart contract code that represent business logic.
- **Querying**: This mode is concerned with simply probing the contract for data and is executed locally on the nodes for performance reasons. Pact uses Lisp-like syntax and represents in the code exactly what will be executed on the blockchain, as it is stored on the blockchain in human-readable format. This is in contrast to Ethereum's EVM, which compiles into bytecode for execution, which makes it difficult to verify what code is in execution on the blockchain. Moreover, it is Turing incomplete, supports immutable variables, and does not allow null values, which improves the overall safety of the transaction code execution.

It is not possible to cover the complete syntax and functions of Pact in this short introduction; however, a small example is shown here that shows the general structure of a smart contract written in Pact. This example shows a simple addition module that defines a function named addition that takes three parameters. When the code is executed, it adds all three values and displays the result:

The following example has been developed using the online Pact compiler available at `https://pact.kadena.io/`.

```
"Define keyset"
(define-keyset 'admin-keyset (read-keyset "admin-keyset"))
"Define module"
(module addition 'admin-keyset
"Define function addition that takes 3 argument of type integer"
  (defun addition (x y z : integer)
"Run format"
    (format "Result : {}"  [( + x ( + y z ) )])
)
)
"Run addition function with three arbitrary numbers to add"
(addition 432 4562 87)
```

Figure 22.3: Pact example

When the code is run, it produces the output shown as follows:

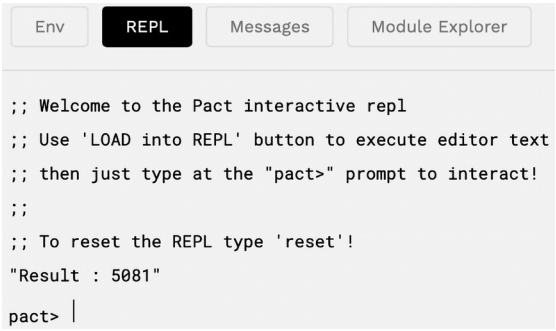

Figure 22.4: The output of the code

As shown in the preceding example, the execution output exactly matches the code layout and structure, which allows greater transparency and limits the possibility of malicious code execution.

Kadena is a new class of blockchain that introduces the novel concept of **pervasive determinism**, where, in addition to standard public/private-key-based data origin security, an additional layer of fully deterministic consensus is also provided. It provides cryptographic security at all layers of the blockchain, including transactions and the consensus layer.

Documentation and source code for Pact can be found at `https://github.com/kadena-io/pact`.

Kadena also introduced a public blockchain in January 2018, which is another leap forward in building blockchains with massive throughput. The novel idea in this proposal is to build a **Proof of Work** (**PoW**) parallel chain architecture. This scheme works by combining individually mined chains on peers into a single network. The result is massive throughput capable of processing more than 10,000 transactions per second.

Various whitepapers on Kadena are available here: `https://docs.kadena.io/basics/whitepapers/overview`.

In the next section, we introduce EOS, which aims to be a scalable blockchain and has introduced several innovative ideas.

EOS

The **EOSIO**, also known as EOS, blockchain was developed by block.one (`https://block.one/`) in the C++ programming language. It was first released on January 31, 2018. It has been built for both public and private blockchain use cases.

A new concept of system resources has been introduced with EOS. Just like the usual concept of computing resources in a computer, such as RAM, CPU, and **Networking** (**NET**), the EOS blockchain uses the same concept of resources to manage the blockchain. The amount of resources allocated is directly proportional to the amount of EOS stake—meaning the higher the stake, the more resources you get. This resource model protects against abuse because in order to game the system, an attacker would need a large amount of stake.

Resources

During the process of staking, users (stakers) specify how much of the stake is going to be allocated for CPU and NET. RAM is not allocated as a result of staking but is required to be bought separately from the RAM market.

We describe these resources one by one in the next sections.

CPU

CPU is required for executing transactions. It is the processing power of an account. It represents the processing time of an action in microseconds. It is referred to as **CPU bandwidth**.

RAM

This is used for data storage on the network. It is required by nodes to store account data in the blockchain state. It is required to be purchased by developers to run applications. Data such as the name of the account, relevant metadata, permissions, token balance, and public keys is stored in RAM. It can be thought of as hard disk space. RAM can also be traded with other accounts for a fee if it's no longer required by an account.

NET

NET is the network bandwidth measured in bytes. This is the amount of bandwidth a user is allowed to use.

CPU and NET combined are also commonly referred to as bandwidth.

Components

EOS consists of a number of components, which we'll describe in the following sections.

nodeos

This is the core element that runs as a **daemon** on every EOSIO blockchain node. It runs a blockchain node, which can also be configured with plugins for additional functionality.

It handles the blockchain persistence layer, P2P networking, and smart contract code scheduling.

cleos

This is the primary command-line tool that is used to interact with `nodeos`. Interaction is done via REST APIs exposed by `nodeos`. **cleos** is also used to test and deploy smart contracts.

keosd

This is the key management daemon responsible for storing private keys and signing messages.

We can visualize this architecture in the following diagram:

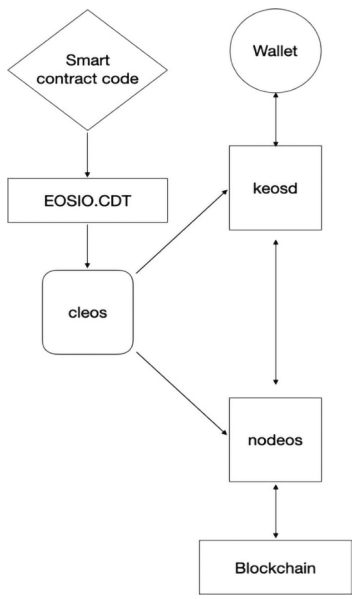

Figure 22.5: EOS high-level architecture

Other components of the EOS blockchain include accounts, transactions and actions, and wallets, which we will introduce next.

Accounts

Accounts in EOS are represented by strings of 12 characters. They act as identifiers on the blockchain. An EOS account also has permissions associated with it that define which actions are allowed to be performed by the account, and are required for actions such as staking, voting, and sending or receiving funds. Accounts are controlled by cryptographic key pairs (a pair of private and public keys). An EOS account name can only contain a-z letters in lowercase, a period (.), and numbers 1-5, and it must start with a letter.

Transactions, actions, and blocks

A **transaction** can be defined as an atomic change to the blockchain, usually as a result of smart contract execution. Transactions make up the bodies of the **blocks**. A block is composed of a **block header** and transactions. One or more actions make up a transaction. An action can be described as a suggested change in the blockchain or a call to a smart contract. There are three main types of actions in EOSIO, namely, *calling actions*, *inline actions*, and *deferred actions*. Calling actions are calls made by users to a contract, inline actions are calls made between different contracts or within the same contract, and deferred actions represent deferred transactions.

Wallet

A wallet is an encrypted file created by a client such as `cleos`. It manages the private keys and transaction signing. A wallet file is unlocked (decrypted) by using a master key. It can be in a locked or unlocked state.

EOS supports the development of smart contracts and provides libraries for developers to use. We explore them in the next section.

Developing with EOS

Development on EOS is made possible by several tools and libraries, which we will describe in this section.

EOSIO.CDT

This is the contract development toolkit. It is a set of tools that enables smart contract development on EOS. It is a toolchain for creating **Web Assembly** (**Wasm**). It contains several elements, such as:

- `eosio-cpp`: Used for compiling contract source code
- `eos-ld`: The web assembly linker for smart contracts
- `eosio-init`: Generates the skeleton code and directory structure for a smart contract
- `eos-abidiff`: Compares two ABI files and outputs the difference

Tezos

Tezos is a generic self-amending cryptographic ledger, which means that it not only allows decentralized consensus on the state of the blockchain but also allows consensus on how the protocol and nodes are evolved over time. Tezos has been developed to address limitations in the Bitcoin protocol (and other similar blockchains) such as contentious issues arising from hard forks, costs, and mining power centralization due to PoW, limited scripting ability, and security issues. It has been developed in a purely functional language called **OCaml**:

- The whitepaper is available at `https://tezos.com/whitepaper.pdf`.
- The position paper is available at `https://tezos.com/position-paper.pdf`.
- The source code is available at `https://gitlab.com/tezos/tezos`.

The architecture of a Tezos distributed ledger is divided into three layers: the network layer, the consensus layer, and the transaction layer. This decomposition allows the protocol to evolve in a decentralized fashion. For this purpose, a generic network shell is implemented in Tezos that is responsible for maintaining the blockchain, which is represented by a combination of the consensus and transaction layers. This shell provides an interface layer between the network and the protocol.

The concept of a **seed protocol** has also been introduced, which is used as a mechanism to allow stakeholders on the network to approve any changes to the protocol.

The Tezos blockchain starts from a seed protocol, in contrast with a traditional blockchain that starts from a genesis block.

This seed protocol is responsible for defining procedures for amendments in the blockchain and even the amendment protocol itself. Originally, the reward mechanism in Tezos was based on a **Proof of Stake** (**PoS**)-based algorithm called Liquid PoS, hence there was no PoW-style mining requirement. However, the protocol was still in the Nakamoto style, meaning probabilistic finality. This resulted in more efficiency and lower energy consumption and unlike PoS, in Liquid PoS, the number of validators was not fixed, which allowed anyone to produce blocks based on the amount of stake they have. However, as the protocol evolved, so did the consensus algorithm. The latest consensus protocol used by Tezos is Tenderbake. It is a classical BFT-style algorithm based on the Tendermint protocol. This means that the finality is now deterministic, and a block is finalized under a minute as compared to probabilistic finality, which took around six minutes in the previously implemented proof of stake protocol called Emmy+.

A domain-specific language called **Michelson** has been developed in Tezos for writing smart contracts, which is a stack-based Turing-complete language. Smart contracts in Tezos are formally verifiable, which allows the code to be mathematically proven for its correctness.

Tezos completed crowdfunding via an ICO of 232 million USD in July 2017. Their public network was released in June 2018.

Tezos code is available at `https://gitlab.com/tezos/tezos`.

Tezos can be distinguished as a platform that is not only a smart contract platform like Ethereum, but also has a built-in governance mechanism and supports the formal verification of contract code. These two additional properties make it a different and arguably better protocol than existing traditional blockchain networks.

Due to the governance mechanism, there is no central control by either developers or miners on the blockchain, and formal verification allows the development of secure and formally verified smart contracts. By allowing formal verification, bugs can be avoided, which leads to better security. Tezos' cryptocurrency is called **tezzies** and is symbolized by the letters XTZ.

Now let's have a look at the Tezos architecture, and see the different elements that make up Tezos and how they fit together.

Architecture

Before discussing the components of a Tezos network, let's first see what a Tezos network is.

Network

The network is the underlying blockchain network using the **Peer-to-Peer** (**P2P**) protocol where all Tezos nodes exist and communicate with each other.

There are two main test networks available for Tezos called Limanet and Mumbainet. More details are available here: `https://tezos.gitlab.io/introduction/test_networks.html`.

Tezos' mainnet is the main production network used by Tezos. More information and a list of block explorers for Tezos are available here: `https://wiki.tezos.com/build/clients/block-explorers`.

The Tezos network consists of several components, which we'll describe in the following sections.

Client

The client is a basic wallet and acts as a primary interface to the node. Clients or other third-party applications communicate with the node via RPC, which uses JSON format and the HTTP protocol.

Node

A node can be defined as a peer on the Tezos P2P network. It is an entity that is responsible for connecting with the Tezos blockchain. A node has a local state and consists of a shell and the protocol. The shell consists of a P2P network layer and a validator. The P2P layer is responsible for communicating with other nodes via the **gossip protocol**. The **economic protocol** is the self-amending element that is responsible for different operations, such as transaction interpretation.

A node propagates blocks and operations such as transactions, accusation, activation, delegation, endorsement, and origination. Let's look at each of these terms:

- A **transaction** in Tezos can be defined as a transfer of funds between two accounts, or a smart contract execution.
- **Accusation** is the process whereby a node can accuse another node of deviating from the protocol or, in other words, abusing or misusing the network, such as injecting incompatible blocks into the blockchain.
- **Activation** is the process of claiming the tokens from the ICO sale and activating the address on the blockchain.
- **Delegation** is the mechanism by which a token holder delegates the rights of baking (baking is Tezos' term for validating transactions and blocks) to another party.
- **Endorsing** is a mechanism whereby a baker is asked to validate and witness a block to check that it has been created correctly and is a valid block. Endorsers are also rewarded with tezzies for their activity.
- **Origination** is the action (or operation) of creating a new smart contract.

All the aforementioned operations are actions that result in a state change in the blockchain.

Other than the peer nodes, there are other daemons, which can be endorsers, bakers, or accusers. We'll discuss these individually, as follows.

Endorser

This is a node on the Tezos blockchain network that has the endorsement right, which results in increasing the endorsed block's score. A block's score is a measure of its weight or fitness to be considered the head of the blockchain. It can be defined as a unit of comparing contexts. Context is simply defined as the state of the blockchain. If there are conflicting blocks, the context's score is measured, and the highest-scoring (or fittest) block becomes the head of the blockchain.

In relation to the context and operations, there is another concept called the **economic protocol**, which has been introduced in Tezos. This is defined as the application that runs on top of the blockchain and defines its state (context) and actions that result in state changes (operations).

Baker

A node is a baker when its role is to add blocks in the Tezos blockchain. During the baking process, a baker picks up transactions accumulated in its memory pool that have been propagated on the network.

A set of consecutive blocks is called a cycle. A cycle represents blocks from a certain height to another height and is used as a perception of time on the blockchain. Another relevant concept is called a roll, which means an amount of tezzies as a unit, to establish a delegate's rights of baking in a cycle.

Baking in Tezos is the equivalent of mining in Bitcoin. In PoW blockchains such as Bitcoin or Ethereum, the right to add a new block to the blockchain is won by solving the PoW problem. In Tezos, this right is assigned randomly to a baker who owns tezzies. One key point to note is that it is not the baker that is selected randomly; it is the token that a baker holds.

The tokens can also be delegated to someone else, and if a token out of those delegated tokens is randomly chosen for baking, then the delegated party will win the right to add a new block.

A baker is required to create a safety deposit if chosen as the next block creator, which ensures the honesty of the baker. If a baker tries to be dishonest, it would be punished and its safety deposit would be lost. If the baker honestly creates a new block that is accepted, it obtains tezzies as a reward.

A list of bakers can be found here: `https://mytezosbaker.com`.

Accuser

An accuser node provides evidence to show that the accused node has attempted to perform some illegal activity. In return for this effort, the accuser is rewarded with some funds from the accused node's baking deposit. Illegal activities primarily include operations where a baker signs two different blocks at the same block height or when more than one endorsement operation is injected by an endorser within the same baking cycle.

Like any blockchain network, an account is required to perform transactions on a blockchain network. We introduce accounts in Tezos next.

Accounts

There are two types of accounts within Tezos, namely **implicit accounts**, which are identified by the prefix `tz1`, and **originated accounts**, which are identified by the `kt1` prefix.

Implicit accounts are created by using a public and private key pair. They have an account owner representing the owner of the private key and the account balance. The public address for these accounts is identified by the `tz{1,2,3}` prefix, which is derived from the public key.

The other type of account is for smart contracts and is called an originated account. These are identified by the prefix `kt1`. They are created by the origination operation from another contract. Originated accounts cannot act as a baker and can be spendable or delegatable. They are managed via an implicit account or another contract.

These accounts have four fields, namely, **manager**, **amount**, **delegatable**, and **delegate**. Manager contains the private key for the account. Amount, as the name suggests, holds the number of tezzies for this account. The delegatable field describes whether this account is capable of delegating baking operations or not. Finally, the delegate field identifies the delegated account for baking.

The high-level architecture of Tezos can be visualized as follows:

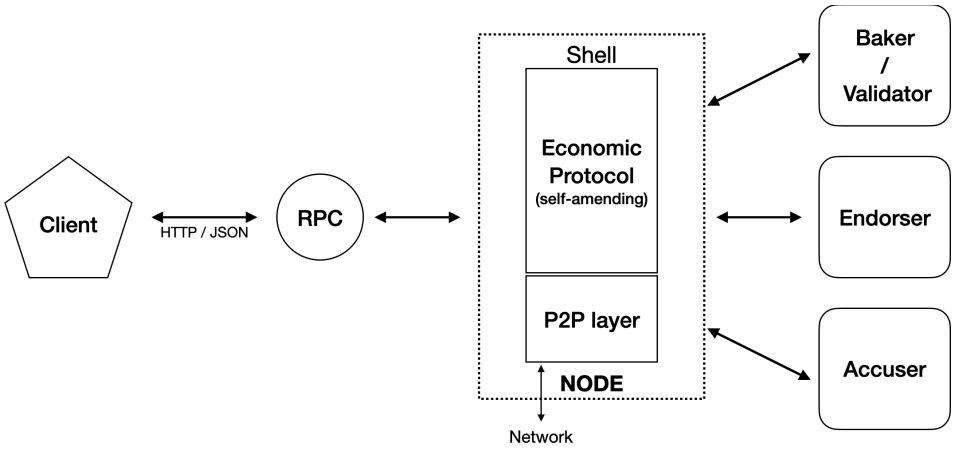

Figure 22.6: Tezos high-level architecture

The Tezos ecosystem also has two scalability solutions, including optimistic rollups and sidechains. Optimistic rollups include a preliminary solution for transaction scalability called **Transaction Optimistic Rollups** (**TORU**) and **Smart Contract Optimistic Rollups** (**SCORU**), which will allow the processing of smart contracts within the rollup using **Web Assembly Virtual Machine** (**WASMVM**). Rollups use a new type of consensus called Tenderbake, which is based on Tendermint. Sidechain-based solutions include the **DEKU chain** (**DEKU-C**), which is a **Proof of Authority** (**PoA**) public chain using a variant of the Tendermint consensus protocol. DEKU-C is based on **DEKU Parametric** (**DEKU-P**), which allows you to create custom permissions on private chains.

Now that we understand the theoretical foundation, let's see what options are available for smart contract development in Tezos.

Development

Smart contracts in Tezos can be readily formally verified, thus increasing the reliability and security of the contracts. A domain-specific language called **Michelson** has been developed for writing smart contracts for Tezos. In contrast with Ethereum's Solidity, which needs to be compiled into bytecode first for it to be executed on the VM, Michelson can directly run normal, human-readable code on the Tezos VM. This approach helps with the formal verification of the smart contract code. Michelson is a stack-based and strongly typed language, which allows smart contracts written using Michelson to be formally verified.

Strongly typed languages have strictly defined restrictions imposed by the compiler of the language, which enforces certain rules around the data types. Usually, these are restrictions on the automatic conversion of one data type to another, and variables are required to have a well-defined type. If these rules are violated, then exceptions are usually raised at compile time.

Smart contracts are identified by addresses starting with `KT1`. Smart contracts are created with an operation called origination, meaning contract registration on the blockchain network.

Some languages other than Michelson that have been developed for smart contract development for Tezos are as follows:

- **LIGO:** This is an easy-to-use smart contract language for Tezos. There are three flavors available for LIGO, namely PascaLIGO, CameLIGO, and ReasonLIGO. More information can be found at `https://ligolang.org`.
- **Fi:** A high-level language for Michelson. More information is available at `https://fi-code.com`.
- **Liquidity**: A high-level typed smart contract language. More information is available at `https://www.liquidity-lang.org`.

There are also various libraries that can be used to integrate Tezos with an application. A selection of these libraries is as follows:

- **ConseilJS**: `https://cryptonomic.github.io/ConseilJS/#/`
- **Taquito**: `https://tezostaquito.io`
- **TezBridge**: `https://docs.tezbridge.com`
- **eztz**: `https://github.com/TezTech/eztz`

Wallets

There are quite a few wallets available for Tezos. Software wallets include Galleon (`https://cryptonomic.tech/galleon.html`), AirGap (`https://airgap.it/`), Kukai (`https://kukai.app/`), and ZenGo (`https://zengo.com/`). Hardware wallets include Ledger (`https://www.ledger.com/`) and Trezor (`https://trezor.io/`).

Moreover, the Tezos client **Command-Line Interface** (**CLI**) can also be used to support basic wallet functionality.

This completes our basic introduction to Tezos. It is a vast subject, and these few pages cannot do justice to the vast and complex Tezos ecosystem. However, this basic introduction should serve as a solid foundation to explore further.

Tezos' official documentation is available at `https://tezos.gitlab.io`.

In the next section, we discuss Ripple, which offers a blockchain platform for global payments.

Ripple

Introduced in 2012, **Ripple** is a currency exchange and real-time gross settlement system.

In Ripple, the payments are settled without any waiting, as opposed to traditional settlement networks, where settlement can take days.

It has a native currency called **Ripples** (**XRP**), but it also supports non-XRP payments. This system is considered similar to a traditional money transfer mechanism known as **Hawala**. This system works by making use of agents who take money and a password from the sender, then contact the payee's agent and instruct them to release funds to the person who can provide the password. The payee then contacts the local agent, tells them the password, and collects the funds. An analogy to the agent is a **gateway** in Ripple. This is just a very simple analogy; the actual protocol is rather complex, but it is the same in principle.

The Ripple network is composed of various nodes that can perform different functions based on their type:

- **User nodes**: These nodes are used in payment transactions and can send or receive payments.
- **Validator nodes**: These nodes participate in the consensus mechanism. Each server maintains a set of unique nodes, which it needs to query while achieving consensus. Nodes in the **Unique Node List** (**UNL**) are trusted by the server involved in the consensus mechanism, which will accept votes only from this list of unique nodes.

Ripple is sometimes not considered a truly decentralized network as there are network operators and regulators involved. However, it can be considered decentralized due to the fact that anyone can become part of the network by running a validator node. Moreover, the consensus process is also decentralized because any proposed changes to the ledger have to be decided by following a scheme of **super majority voting**. However, this is a hot topic among researchers and enthusiasts and there are arguments against and in favor of each school of thought. There are some discussions online that readers can refer to for further exploration of these ideas. You can find a couple of these online discussions at the following addresses:

- `https://financefeeds.com/sec-claims-xrp-is-a-security-due-to-centralized-nature/`
- `https://thenextweb.com/hardfork/2018/02/06/ripple-report-bitmex-centralized/`
- `https://cryptoslate.com/vitalik-buterin-says-xrp-is-completely-centralized-draws-ripple-ctos-reaction/`

Ripple maintains a globally distributed ledger of all transactions that are governed by a novel low-latency consensus algorithm called the **Ripple Protocol Consensus Algorithm** (**RPCA**). The consensus process works by achieving agreement on the state of an open ledger containing transactions by seeking verification and acceptance from validating servers in an iterative manner until an adequate number of votes is collected. Once enough votes are received (a super majority, initially 50% and gradually increasing with each iteration up to at least 80%), the changes are validated and the ledger is closed. At this point, an alert is sent to the whole network indicating that the ledger is closed.

The original research paper for RPCA is available at `https://ripple.com/files/ripple_consensus_whitepaper.pdf`.

Another updated version is available at `https://arxiv.org/ pdf/1802.07242.pdf`.

In summary, the consensus protocol is a three-phase process:

1. **Collection phase**: In this phase, validating nodes gather all transactions broadcast on the network by account owners and validate them. Transactions, once they are accepted, are called *candidate transactions* and can be accepted or rejected based on the validation criteria.
2. **Consensus phase**: After the collection phase, the consensus process starts, and after achieving it, the ledger is closed.
3. **Ledger closing phase**: This process runs asynchronously every few seconds in rounds, and when it completes, the ledger is opened and closed (updated) accordingly:

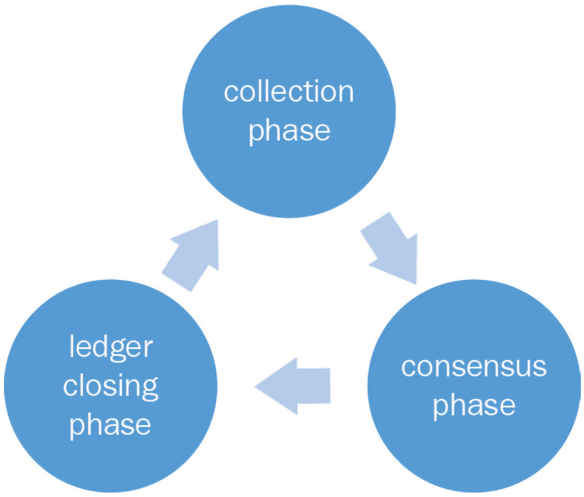

Figure 22.7: Ripple consensus protocol phases

In a Ripple network, there are a number of components that work together in order to achieve consensus and form a payment network. These components are discussed individually here:

- **Server**: This component serves as a participant in the consensus protocol. Ripple server software is required in order to be able to participate in the consensus protocol.
- **Ledger**: This is the main record of balances of all accounts on the network. A ledger contains various elements, such as the ledger number, account settings, transactions, timestamp, and a flag that indicates the validity of the ledger.
- **Last closed ledger**: A ledger is closed once consensus is achieved by validating nodes.
- **Open ledger**: This is a ledger that has not been validated yet and no consensus has been reached about its state. Each node has its own open ledger, which contains proposed transactions.
- **Unique node list**: This is a list of unique trusted nodes that a validating server uses in order to seek votes and subsequent consensus.
- **Proposer**: As the name suggests, this component proposes new transactions to be included in the consensus process. It is usually a subset of nodes (UNL defined in the previous point) that can propose transactions to the validating server.

Like any other blockchain, a fundamental activity in Ripple is the transaction. We introduce the design and architecture of transactions in Ripple in the next section.

Transactions

Transactions are created by the network users in order to update the ledger. A transaction is expected to be digitally signed and valid in order for it to be considered as a candidate in the consensus process. Each transaction costs a small amount of XRP, which serves as a protection mechanism against **Denial of Service** (**DoS**) attacks caused by spamming.

There are different types of transactions in the Ripple network. A single field within the Ripple transaction data structure called `TransactionType` is used to represent the type of the transaction. Transactions are executed by using a four-step process:

1. First, transactions are prepared whereby an unsigned transaction is created by following the standards.
2. The second step is signing, where the transaction is digitally signed by the sender to authorize it.
3. After this, the actual submission to the network occurs via the connected server.
4. Finally, the verification is performed to ensure that the transaction is validated successfully.

A transaction in Ripple is composed of various fields that are common to all transaction types. These fields are listed as follows with a description:

- `Account`: This is the address of the initiator of the transaction.
- `AccountTxnID`: This is an optional field that contains the hash of another transaction. It is used to chain the transactions together.
- `Fee`: This is the amount of XRP.
- `Flags`: This is an optional field specifying the flags for the transaction.
- `LastLedgerSequence`: This is the highest sequence number of the ledger in which the transaction can appear.
- `Memos`: This represents optional arbitrary information.
- `SigningPubKey`: This represents the public key.
- `Signers`: This represents signers in a `multisig` transaction.
- `SourceTag`: This represents either the sender of, or the reason for, the transaction.
- `TxnSignature`: This is the verification digital signature for the transaction.

Roughly, transactions can be categorized into three types, namely **payments-related**, **order-related**, and **account-and-security-related**. These types and their unique fields are described in the following sections.

Payments-related

There are several fields in this category that result in certain actions. These fields are described as follows:

- `Payment`: This transaction is most commonly used and allows one user to send funds to another.

- `PaymentChannelClaim`: This is used to claim XRP from a payment channel. A payment channel is a mechanism that allows recurring and unidirectional payments between parties. This can also be used to set the expiration time of the payment channel.
- `PaymentChannelCreate`: This transaction creates a new payment channel and adds XRP to it in drops. A single drop is equivalent to 0.000001 of an XRP.
- `PaymentChannelFund`: This transaction is used to add more funds to an existing channel. Similar to the `PaymentChannelClaim` transaction, this can also be used to modify the expiration time of the payment channel.

Order-related

This type of transaction includes the following two fields:

- `OfferCreate`: This transaction represents a limit order, which represents an intent for the exchange of currency. It results in creating an offer node in the consensus ledger if it cannot be completely fulfilled.
- `OfferCancel`: This is used to remove a previously created offer node from the consensus ledger, indicating withdrawal of the order.

Account-and security-related

This type of transaction includes the fields listed as follows. Each field is responsible for performing a certain function:

- `AccountSet`: This transaction is used to modify the attributes of an account in the Ripple consensus ledger.
- `SetRegularKey`: This is used to change or set the transaction signing key for an account. An account is identified using a Base58 Ripple address derived from the account's master public key.
- `SignerListSet`: This can be used to create a set of signers for use in multisignature transactions.
- `TrustSet`: This is used to create or modify a trust line between accounts.

Interledger

Original work on the **Interledger** protocol was started by Ryan Fugger in 2004. With the introduction of Bitcoin in 2009, this work generated even more interest, and since then many contributions have been made to this project.

A protocol for Interledger payments was invented by Stefan Thomas and Evan Schwartz from Ripple. The research paper is available at `https://interledger.org/interledger.pdf`.

Interledger is an open protocol suite for cross-ledger payments. It is composed of four layers: **Application**, **Transport**, **Interledger**, and **Ledger**. Each layer is responsible for performing various functions under certain protocols. These functions and protocols are described in the following section.

The official website is `https://interledger.org`.

The RFCs of this protocol are available at `https://github.com/interledger/rfcs`.

Application layer

Protocols running on this layer govern the key attributes of a payment transaction. Examples of application layer protocols include **Simple Payment Setup Protocol** (**SPSP**) and **Open Web Payment Scheme** (**OWPS**). SPSP is an Interledger protocol that allows secure payment across different ledgers by creating connectors between them. OWPS is another scheme that allows consumer payments across different networks.

Once the protocols on this layer have run successfully, protocols from the transport layer will be invoked in order to start the payment process.

Transport layer

This layer is responsible for managing transactions. Protocols such as **Optimistic Transport Protocol** (**OTP**), **Universal Transport Protocol** (**UTP**), and **Atomic Transport Protocol** (**ATP**) are available currently for this layer. OTP is the simplest protocol, which manages payment transfers without any escrow protection, whereas UTP provides escrow protection. ATP is the most advanced protocol, which not only provides an escrowed transfer mechanism but also makes use of trusted notaries to further secure transactions.

Interledger layer

This layer provides interoperability and routing services. This layer contains protocols such as **Interledger Protocol** (**ILP**), **Interledger Quoting Protocol** (**ILQP**), and **Interledger Control Protocol** (**ILCP**). The ILP packet provides the final target (destination) of the transaction in a transfer.

ILQP is used in making quote requests by the senders before the actual transfer. ILCP is used to exchange data related to routing information and payment errors between connectors on the payment network.

Ledger layer

This layer contains protocols that enable the communication and execution of payment transactions between connectors. **Connectors** are objects that implement the protocol for forwarding payments between different ledgers. The ledger layer can support various protocols such as **Simple Ledger Protocol** (**SLP**), various blockchain protocols, legacy protocols, and different proprietary protocols.

Ripple Connect consists of various **plug-and-play** modules that allow connectivity between ledgers by using the ILP.

Plug and play is a feature of software or electronic devices that allows them to work the first time they are used without requiring any configuration by the user.

It enables the exchange of required data between parties before the transaction, visibility, fee management, delivery confirmation, and secure communication using transport layer security. A third-party application can connect to the Ripple network via various connectors that forward payments between different ledgers.

All layers described in the preceding sections make up the architecture of the ILP. Overall, Ripple is a solution that targets the financial industry and makes real-time payments possible without any settlement risk.

As this is a very feature-rich platform, covering all aspects of it is not possible in this brief introduction. However, Ripple documentation is available at `https://ripple.com/`.

In the next section, we'll discuss another payment network called Stellar, which emerged about two years after the introduction of Ripple.

Stellar

Stellar is a payment network based on blockchain technology and a novel consensus model called **Federated Byzantine Agreement** (**FBA**). FBA works by creating quorums of trusted parties. **Stellar Consensus Protocol** (**SCP**) is an implementation of FBA.

Stellar Consensus Protocol

Some of the key issues identified in the Stellar white paper are the cost and complexity of the current financial infrastructure. This limitation warrants the need for a global financial network that addresses these issues without compromising the integrity and security of the financial transaction. This requirement resulted in the invention of SCP, which is a demonstrably safe consensus mechanism.

The original research paper for SCP is available at `https://www.stellar.org/papers/stellar-consensus-protocol.pdf`.

SCP has four main properties, which are described here:

- **Decentralized control**: This allows participation by anyone without any central party.
- **Low latency**: This addresses the much-desired requirement of fast transaction processing.
- **Flexible trust**: This allows users to choose which parties they trust for a specific purpose.

- **Asymptotic security**: This makes use of digital signatures and hash functions for providing the required level of security on the network.

The Stellar network allows the transfer and representation of the value of an asset by its native digital currency, called **lumens**, abbreviated as XLM. Lumens are consumed when a transaction is broadcast on the network, which also serves as a deterrent against DoS attacks.

At its core, the Stellar network maintains a distributed ledger that records every transaction and is replicated on each Stellar server (node). The consensus is achieved by verifying transactions between servers and updating the ledger with updates. The Stellar ledger can also act as a distributed exchange order book by allowing users to store their offers to buy or sell currencies.

The core software is available at `https://github.com/stellar/stellar-core`.

With this, we complete our introduction to Stellar. Next, we introduce Rootstock, which is a separate blockchain attached to the Bitcoin blockchain using a two-way peg and introduces smart contract functionality to the Bitcoin ecosystem.

Rootstock

Before discussing **Rootstock** (**RSK**) in detail, it's important to define and introduce some concepts that are fundamental to its design. These concepts include **two-way pegging**, **sidechains**, and **drivechains**.

Two-way pegging

This is a mechanism by which a value (coins) can be transferred between one blockchain and another. There is no real transfer of coins between chains. The idea revolves around the concept of locking the same amount and value of coins in a Bitcoin blockchain (the main chain) and unlocking the equivalent number of tokens in the secondary chain.

Bearing this definition in mind, sidechains will be defined next.

Sidechain

This is a blockchain that runs in parallel with a main blockchain and allows the transfer of values between them. This means that tokens from one blockchain can be used in the sidechain and vice versa. This is also called a pegged sidechain because it supports two-way pegged assets.

The concept of the sidechain was originally developed by Blockstream.

Blockstream's website is located at `https:// blockstream.com`.

Drivechain

This is a relatively new concept, where control of unlocking the locked bitcoins (in the main chain) is given to the miners who can vote on when to unlock them. This is in contrast with sidechains, where consensus is validated through a simple payment verification mechanism in order to transfer the coins back to the main chain.

For more details regarding drivechains, sidechains, and two-way peg designs, refer to the original research paper at `https://docs.rsk.co/ Drivechains_Sidechains_and_Hybrid_2-way_peg_Designs_R9.pdf`.

Now that we understand the basic ideas behind these core concepts, let's discuss RSK in more detail.

RSK is a smart contract platform that has a two-way peg into the Bitcoin blockchain. The core idea is to increase the scalability and performance of the Bitcoin system and enable it to work with smart contracts. RSK runs a Turing-complete deterministic VM called the **Rootstock Virtual Machine** (**RVM**).

It is also compatible with the EVM and allows Solidity-compiled contracts to run on RSK. Smart contracts can also run under the time-tested security of Bitcoin. The RSK blockchain works by merge-mining (merge-mining is the process of mining more than one blockchain at once) with Bitcoin. This allows RSK to achieve the same level of security as Bitcoin. This is especially true for preventing double-spends and achieving settlement finality. It allows scalability, up to 400 transactions per second due to faster block times and other design considerations.

The research paper is available at `https://uploads.strikinglycdn.com/files/ec5278f8-218c-407a-af3c-ab71a910246d/RSK%20White%20Paper%20-%20Overview.pdf` should you want to explore it further.

RSK has released a mainnet called Bamboo, which is currently in beta.

Further information on Rootstock is available at `http://www.rsk.co/`.

In the next section, we introduce some projects related to decentralized storage.

Solana

Solana is a layer 1 blockchain platform that was launched in 2018 with a focus on speed, security, scalability, and decentralization. It has smart contract support and is currently in the beta stage with growing popularity. Although it is an operational network with some production systems, there are still some technical issues being addressed. The ledger uses a verifiable delay function where time is treated as data and supports millions of nodes with the help of GPUs. The native token on the platform, the SOL coin, is used for governance and incentivization.

Solana's main innovations include:

- **Proof of History** (**PoH**) for ordering events.
- The Tower BFT consensus algorithm for voting and fork selection.

- Turbine for efficient block propagation by breaking blocks into smaller 64 KB packets and streaming them over UDP through a random path implemented per packet through the network.
- Parallel execution of smart contracts through a runtime called Sealevel. Parallel execution is made possible because Solana transactions preemptively know all states it will read or write, which allows for the concurrent execution of transactions.
- Optimized transaction validation through a pipelined transaction processing unit.
- A horizontally scalable database called Cloudbreak, which leverages memory-mapped files and sequential operations to gain speed and efficiency.
- Archivers for distributed ledger storage to store ledger data.

Note that PoH is not a consensus algorithm, but it enables the ordering of events using a cryptographic clock and leads to consensus. Tower BFT is derived from PBFT and PoH reduces message complexity in a BFT protocol, resulting in high throughput and sub-second finality times.

Solana uses proof of stake and Tower BFT as its consensus algorithms. PoH is a key innovation that allows for a self-consistent record of events by proving the order and passage of time between events without relying on an external source. This leads to reduced communication complexity and improved performance.

Proof of History

Note that time in distributed systems is crucial. If time is synchronized among processes, i.e., a synchronized clock is available in a distributed network, then communication complexity can be reduced, which results in improved performance. In addition, a node can deduce information from past events instead of asking another node repeatedly about some information. For example, with the availability of a global clock where all nodes are synchronized, the system can establish a system-wide history of events. For example, a timestamp on an event can inform a node when this event occurred on the globally synchronized time across the network instead of asking a node who produced that event when this event occurred.

Another application of a synchronized clock is that entities in the system can deduce if something has expired; e.g., a timestamped security token can immediately tell a node how much time has elapsed since its creation. The node can infer if it is valid anymore or not and not something that occurred in the distant past, making this token no longer applicable and expired.

In replication protocols, clock synchronization also plays a crucial role. If nodes have synchronized clocks, that can lead to consistency because every node will have the same view of the order of events.

The system can only establish a global notion of time and history if the time is synchronized among nodes. It is usually possible in practical systems using the NTP protocol.

So far, we have established that synchronized time is a valuable construct in distributed systems for performance gains. In other words, if we can replace communication with local computation, we can gain tremendous efficiency.

Also, synchrony and time together solve consensus quickly and easily. Safety and liveness are the two fundamental requirements and are easy to implement with a trusted clock and synchronous network.

However, networks are empirically asynchronous. We also know that a trusted synchronized clock in distributed networks is difficult to maintain. Blockchain and distributed systems are characterized by no clocks, making them slow due to inherent asynchrony and the need for complex message passing for ordering events and agreements.

On the other hand, a reliable, trusted clock makes network synchronization much simpler and quicker, which leads to high-speed networks. Solana's PoH allows the system to keep time reliably between non-trusting computers. In short, PoH enables clocks in clockless blockchains.

Solana's PoH is a way to establish the history and provide that global notion of synchronized time among nodes in a distributed network. The key innovation here is that it does not use any external source of time and synchronize nodes using that, e.g., via the NTP protocol; instead, it uses a cryptographic proof to show that some time has passed, and other nodes directly accept this history of events due to cryptographic guarantees. So, instead of relying on a global time source, this mechanism is built into validators that generate a sequence of events with proof of when an event occurred. The following is an explanation of how it works.

In a blockchain network, the right to add a new block is won after solving a puzzle, i.e., PoW, which takes a long time. Although this mechanism is secure and thwarts Sybil attacks (as seen in *Chapter 5*), it is slow. Furthermore, in some chains, a BFT-style consensus is used. In that case, the leader validator who proposes a block only gets to commit after at least two sequential phases, which is also time consuming even in normal conditions. Under failure conditions, it can further slow down with new leader selection (election) and view changes. What if, somehow, there is a deterministic leader election algorithm that can select leaders in quick succession, and each leader quickly proposes new blocks? Then, the algorithm moves to the next leader, and so on. All this without going through a complex leader election process, acknowledgment from other nodes, and running multiple phases to reach a consensus. The problem here is that it's quick to create a deterministic algorithm that can select the next leader, but how do we ensure that what the alogrithm proposes is correct and that the selected leaders are not malicious and do not censor transactions or exhibit other malicious behaviors?

This is where PoH comes in. In Solana, one leader at a time processes transactions and updates the state. Other validators read the state and send votes to the leader to confirm them. This activity is split into very short successive sessions where one leader after another performs this. The ledger is divided into small intervals. These small intervals are 400 ms each. The leader rotation schedule is predetermined and generated randomly based on factors such as the stake and execution of previous transactions. But how can we ensure that the leader rotation is done at the right time and malicious actors cannot skip a leader's turn or censor transactions?

In PoH, the passage of time is proven by creating a sequence of these hashes as shown in *Figure 22.8*:

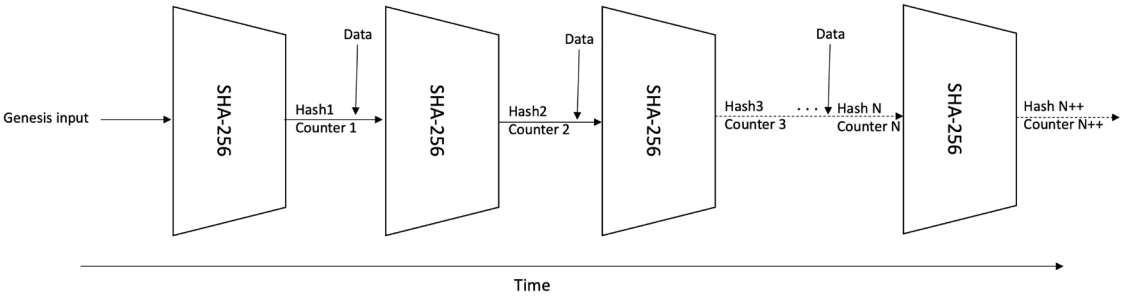

Figure 22.8: Solana PoH sequence

Here, in the figure, a sequence of hash operations is shown. The genesis input (shown at the left in the diagram) is first provided to the hash function. In the next iteration, the output of the previous hash function is used as input to the hash function, and this process continues indefinitely. This sequence is generated using the SHA256 function on a single core. This process cannot parallelize it because the output of the previous hash function can only be known if and only if the hash function has processed the previous input. It is assumed that the functions are cryptographic hash functions that are pre-image resistant. Therefore, this is a purely sequential function. However, this sequence can be verified in parallel using multicore GPUs. As all the inputs and outputs are available, it becomes just a matter of verifying each output, which GPUs can do in parallel. This property makes this sequence a **Verifiable Delay Function** (**VDF**) because the time taken (i.e., delay) in generating the hash sequence can be verified using quick parallel verification. However, there is some debate between cryptographic VDFs introduced by researchers at Stanford and hardware VDFs introduced by Solana researchers.

We can then sample this sequence at regular intervals to provide a notion of the passage of time. This is so because hash generation takes some CPU time (roughly 1.75 cycles for SHA25 instruction on an Intel or AMD CPU), and this process is purely sequential. We can infer from this sequence that some time has passed because generating hashes takes time. Therefore, looking at this sequence from the first to the most recent hash, we can view this sequence of hashes as a representation of passing time.

Since the algorithm generated the first hash in the sequence, some time has been taken up by the hash generation process for each hash that comes after it until the latest hash in the series, meaning some time has passed since the first hash was generated. Furthermore, if we can also add some data with the input hash to the hash function, then we can also deduce that this data must have existed before the next hash and after the previous hash. Thus, this sequence of hashes becomes PoH, proving cryptographically that some events, such as event *e*, occurred before event *f* and after event *d*.

It is a sequential process that runs SHA256 repeatedly and continuously, using its previous output as input. It periodically records a counter for each output sample, e.g., records every 1 second the current state (hash output), similar to a clock ticking. Looking at this structure of sampled hashes at regular intervals, we can infer that some time has passed. It is impossible to parallelize because the previous output is the input for the next iteration.

For example, we can say time passed between counter 1 and counter N (*Figure 22.8*), where time is the SHA256 counter. We can approximate real time from this count. We can also associate some data, which we can append to the input of the hash function; once hashed, we can be sure that data must have existed before the hash is generated. This structure can only be generated in sequence; however, we can verify it in parallel. For example, if 4,000 samples took 40 seconds to produce, it will take only 1 second to verify the entire data structure with a 4,000-core GPU.

The key idea is that PoH transactional throughput is separated from consensus, which is key to scaling. Note that the order of events generated, i.e., the sequence, is not globally unique. Therefore, a consensus mechanism is needed to ascertain the true chain, as anyone can generate an alternate history.

PoH is a cryptographically proven way of saying that time has elapsed. It can be seen as an application-specific verifiable delay function. It encodes the passage of time as data using SHA-256 hashing to hash the incoming events and transactions. It produces a unique hash and count of each event, producing a verifiable ordering of events as a function of time. This means that the time and ordering of events can be agreed upon without waiting to hear from other nodes. In other words, there is no weak subjectivity where nodes must rely on other nodes to determine the system's current state. This results in high throughput because the information that is usually required to be provided by other nodes is already there in the sequence generated by the PoH mechanism and is cryptographically verifiable, ensuring integrity. This means that the protocol can enforce a global order without going through a communication-wise complex agreement protocol or trusting an external source of time for clock synchronization. In summary, instead of trusting the timestamp, PoH allows the creation of a historical record proving that an event *e* occurred at a particular point in time *t*, before another event *f* and after an event *d*. In summary, two key benefits of PoH are:

- Provides a means for trustless ordering of events
- It increases the block production frequency significantly because nodes can select a new block proposer leader without communicating

PoH leadership can be switched without needing to communicate with other nodes, resulting in increased block production frequency. PoH results in high node scalability and low communication complexity compared to BFT-style protocols with high communication complexity and limited node capacity. It provides global read consistency and a cryptographically verifiable passage of time between two events. With PoH, nodes can trust the ordering and timing of events, even before the consensus stage is reached. In other words, it's a "clock before the consensus" approach. The consensus then works by voting on different branches, where nodes vote on a branch they believe is the main chain. Over time, by keeping voting on the chain they first voted on and not voting on another branch, they earn rewards and, eventually, the other branch orphans out.

For voting purposes, Tower BFT is used, which is a variant of PBFT. It is a voting and fork selection algorithm. It is used to vote on the chains produced by PoH to select the true canonical chain. It is less complex communication wise because PoH has already provided an order, and now a decision is only required on the choice of canonical chain. Moreover, PoH provides the timing of events before consensus initiates, and Tower BFT is then used for voting on the canonical chain.

Tower BFT is byzantine fault tolerant because once two thirds of validators have voted on a chain (hash), it cannot be rolled back. Validators vote on a PoH hash for two reasons; first, to decide that the ledger is valid until the hash for which they are voting, i.e., a point in time, and second, to support a fork at a given height, as many forks can exist at a given chain height. Furthermore, PoS in Solana is used for economics and governance to control slashing, inflation, supply, and penalties.

Storage layer blockchain projects

In this section, we introduce some projects that use the principles of blockchain technology to provide decentralized storage systems that can be used by blockchains or by any other project that wishes to use decentralized storage. These projects include Storj, MaidSafe, and BigchainDB.

Storj

Existing models for cloud-based storage are all centralized solutions, which may or may not be as secure as users expect them to be. We need cloud storage systems that are secure, highly available, and above all decentralized. **Storj** aims to provide blockchain-based, decentralized, and distributed storage. It is a cloud shared by the community instead of a central organization.

It allows the execution of storage contracts between nodes that act as autonomous agents. These agents (nodes) execute various functions such as data transfer, validation, and data integrity checks.

Storj's core concept is based on a **Distributed Hash Table** (**DHT**) called **Kademlia**. However, Storj's protocol has been enhanced by adding new message types and functionalities. It also implements a P2P **publish/subscribe** (**pub/sub**) mechanism known as **Quasar**, which ensures that messages successfully reach the nodes that are interested in storage contracts. This is achieved via a storage contract parameter selection mechanism based on Bloom filters.

Storj stores files in an encrypted format spread across the network. Before the file is stored on the network, it is encrypted using AES-256-CTR symmetric encryption and is then stored piece by piece in a distributed manner on the network. This process of dissecting the file into pieces is called **sharding** and results in the increased availability, security, performance, and privacy of the network. Also, if a node fails, the shard is still available because, by default, a single shard is stored at three different locations on the network.

It maintains a blockchain, which serves as a shared ledger and implements standard security features such as public/private key cryptography and hash functions, similar to any other blockchain. As the system is based on hard drive sharing between peers, anyone can contribute by sharing the extra space on their drive and get paid with Storj's own cryptocurrency, called **Storjcoin X** (**SJCX**). SJCX was developed as a **Counterparty** asset and makes use of the Bitcoin blockchain-based Counterparty platform for transactions. This has now been migrated to Ethereum.

A detailed discussion is available at `https://blog.storj.io/post/158740607128/migration-from-counterparty-to-ethereum`.

The Storj code is available at `https://github.com/Storj/`.

MaidSafe

This is another distributed storage system similar to Storj. Users are paid in **SafeCoin** for their storage space contributions to the network. This mechanism of payment is governed by **proof of resource**, which ensures that the disk space committed by a user to the network is available; if not, then the payment of SafeCoin will drop accordingly. The files are encrypted and divided into small portions before being transmitted onto the network for storage.

Another concept of **opportunistic caching** has been introduced with **MaidSafe**, which is a mechanism to create copies of frequently accessed data physically closer to where the access requests are coming from, which results in the high performance of the network. Another novel feature of MaidSafe's network (the SAFE network) is that it automatically removes any duplicate data on the network, thus resulting in reduced storage requirements.

Moreover, the concept of **churning** has also been introduced, which basically means that data is constantly moved across the network so that the data cannot be targeted by malicious adversaries. Churning also keeps multiple copies of data across the network to provide redundancy in case a node goes offline or fails.

Other platforms

This section will briefly consider some novel platforms that have introduced some innovative ideas to blockchain technology.

MultiChain

MultiChain has been developed as a platform for the development and deployment of private blockchains. It is based on Bitcoin code and addresses security, scalability, and privacy issues. It is a highly configurable blockchain platform that allows users to set different blockchain parameters. It supports control and privacy via a granular permissioning layer. The installation of MultiChain is very quick.

To install MultiChain, this link can be followed: `http://www.multichain.com/download-install/`.

Tendermint

Tendermint is software that provides a BFT consensus mechanism and state machine replication functionality to an application. Its main motivation is to develop a general-purpose, secure, and high-performance replicated state machine.

There are two components of Tendermint, which are described in the following sections.

Tendermint Core

This is a consensus engine that enables the secure replication of transactions on each node in the network.

Tendermint Socket Protocol

Tendermint Socket Protocol (**TMSP**) is an application interface protocol that allows interfacing with any programming language to process transactions. Tendermint allows for the decoupling of the application consensus processes, which allows any application to benefit from the consensus mechanism.

Tendermint consensus algorithm

The Tendermint consensus algorithm is a round-based mechanism where validator nodes propose new blocks in each round. A locking mechanism is used to ensure protection against a scenario where two different blocks are selected to be committed at the same height of the blockchain. Each validator node maintains a full locally replicated ledger of blocks that contain transactions. Each block contains a header, which consists of the previous block hash, the timestamp of the proposal of the block, the current block height, and the Merkle root hash of all transactions present in the block.

Cosmos

Tendermint has recently been used in **Cosmos** (`https://cosmos.network`), which is a network of blockchains that allows interoperability between different chains running on the BFT consensus algorithm. Blockchains on this network are called zones. The first zone in Cosmos is called Cosmos Hub, which is, in fact, a public blockchain and is responsible for providing connectivity services to other blockchains. For this purpose, the hub makes use of the **Inter Blockchain Communication** (**IBC**) protocol. The IBC protocol supports two types of transactions called `IBCBlockCimmitTx` and `IBCPacketTx`. The first type is used to provide proof of the most recent block hash in a blockchain to any party, whereas the latter type is used to provide data origin authentication. A packet from one blockchain to another is published by first posting a proof to the target chain. The receiving (target) chain checks this proof in order to verify that the sending chain has indeed published the packet. In addition, Cosmos has its own native currency called Atom. This scheme addresses scalability and interoperability issues by allowing multiple blockchains to connect to the hub.

Tendermint is available at `https://tendermint.com/`.

In this vast ecosystem of blockchains and networks, several initiatives have been taken to introduce more feature-rich platforms with better security, interoperability, developer tools, and toolkits.

Summary

This brings our introduction to some innovative blockchain-based platform services to an end. We started with an introduction to alternative blockchains, which was divided into two main sections, one discussing blockchains and the other assorted platforms and tools. Blockchain technology is a thriving area; as such, changes are quite rapid in existing solutions and new technologies and tools are being introduced almost every day. In this chapter, a careful selection of platforms and blockchains was introduced. New blockchains such as Solana, various protocols such as Ripple and Kadena, and concepts such as sidechains and drivechains were also discussed.

The material covered is intended to provide a strong foundation for more in-depth research into areas that readers are interested in. As said before, blockchain is a very fast-moving field, and there are many other blockchain projects. A simple internet search would reveal many projects. Readers are encouraged to keep an eye on any developments in this field to keep themselves up to date with advancements in this rapidly growing area, but a few were discussed in the chapter to give an idea of current projects.

Join us on Discord!

To join the Discord community for this book – where you can share feedback, ask questions to the author, and learn about new releases – follow the QR code below:

`https://packt.link/ips2H`