TRON: Advanced Decentralized Blockchain Platform

Whitepaper Version: 2.1
TRON Protocol Version: 4.8.0

Contents

1 Introduction **4**
- 1.1 Vision . . . 4
- 1.2 Background . . . 4
- 1.3 History . . . 4
  - 1.3.1 The Genesis of a Decentralized Internet (2017) . . . 4
  - 1.3.2 Developing the Core (2018) . . . 4
  - 1.3.3 Expanding Functionality and Adoption (2019 - 2021) . . . 5
  - 1.3.4 Sustained Growth and Ecosystem Dominance (2022 - Present) . . . 5
  - 1.3.5 Looking Ahead . . . 5
- 1.4 Terminology . . . 5

2 Architecture **6**
- 2.1 Core Layer . . . 6
- 2.2 Storage Layer . . . 6
  - 2.2.1 Blockchain Storage . . . 6
  - 2.2.2 State Storage . . . 6
- 2.3 Application Layer . . . 8
- 2.4 Protocol . . . 8
  - 2.4.1 Protocol Buffers . . . 8
  - 2.4.2 HTTP . . . 8
- 2.5 TRON Virtual Machine (TVM) . . . 8

3 Consensus **9**
- 3.1 Delegated Proof of Stake (DPoS) . . . 9

4 Account **11**
4.1 Types . . . 11
4.2 Creation . . . 11
4.3 Structure . . . 11

5 Block **13**
5.1 Block Header . . . 13
5.1.1 Raw Data . . . 13
5.1.2 Witness Signature . . . 13
5.1.3 Block ID . . . 13
5.2 Transaction . . . 14
5.2.1 Signing . . . 14
5.2.2 Bandwidth Model . . . 14
5.2.3 Fee . . . 14
5.2.4 Transaction as Proof of Stake (TaPoS) . . . 15
5.2.5 Transaction Confirmation . . . 15
5.2.6 Structure . . . 15

6 TRON Virtual Machine (TVM) **17**
6.1 Introduction . . . 17
6.2 Workflow . . . 17
6.3 Performance . . . 17
6.3.1 Lightweight Architecture . . . 17
6.3.2 Robust . . . 17
6.3.3 High Compatibility . . . 17
6.3.4 Low Cost . . . 17

7 Smart Contract **19**
7.1 Introduction . . . 19
7.2 Energy Model . . . 19
7.3 Deployment . . . 19
7.4 Trigger Function . . . 20
7.5 TRON Solidity . . . 20

8 Token **21**
8.1 TRC-10 Token . . . 21
8.2 TRC-20 Token . . . 21
8.3 Beyond . . . 21

9 Governance **22**
9.1 Super Representative . . . 22
9.1.1 General . . . 22
9.1.2 Election . . . 22
9.1.3 Reward . . . 22
9.2 Committee . . . 23
9.2.1 General . . . 23
9.2.2 Dynamic Network Parameters . . . 23
9.2.3 Create Proposal . . . 23
9.2.4 Vote Proposal . . . 24
9.2.5 Cancel Proposal . . . 24
9.3 Structure . . . 24

10 DApp Development **25**
10.1 APIs . . . 25
10.2 Networks . . . 25
10.3 Resources . . . 25

11 Conclusion **26**

Appendix A: Terminology **27**

Appendix B: Dynamic Network Parameters **30**

1 Introduction

1.1 Vision

TRON is an ambitious project dedicated to the establishment of a truly decentralized Internet and its infrastructure. The TRON Protocol, one of the largest blockchain-based operating systems in the world, offers public blockchain support of high throughput, high scalability, and high availability for all Decentralized Applications (DApps) in the TRON ecosystem. Since its inception, TRON has consistently expanded its capabilities, achieving significant adoption in areas such as stablecoin transfers and cultivating a thriving community, further solidifying its prominent position within the blockchain industry.

1.2 Background

The introduction of Bitcoin in 2009 revolutionized society's perception of the traditional financial system in the wake of the Great Recession (2007-2008). As centralized hedge funds and banks collapsed from speculation in opaque financial derivatives, blockchain technology provided a transparent universal ledger from which anybody could glean transaction information. Transactions were cryptographically secured using a Proof of Work (PoW) consensus mechanism, thus preventing double-spending issues.

In late 2013, the Ethereum whitepaper proposed a network in which smart contracts and a Turing-complete Ethereum Virtual Machine (EVM) would allow developers to interact with the network through DApps. However, as transaction volumes in Bitcoin and Ethereum peaked in 2017, it was apparent from the low transaction throughput times and high transaction fees that cryptocurrencies like Bitcoin and Ethereum in their existing state were not scalable for widespread adoption. Thus, TRON was founded and envisioned as an innovative solution to these pressing scalability challenges.

1.3 History

1.3.1 The Genesis of a Decentralized Internet (2017)

TRON's journey began in July 2017 with its establishment in Singapore, driven by a core vision: the creation of a truly decentralized internet. This ambitious undertaking was quickly supported by a successful Initial Coin Offering (ICO) in August 2017, securing crucial development capital. An early commitment to transparency and open collaboration was demonstrated with the launch of its open-source protocol in December 2017, inviting community participation from its inception[1].

1.3.2 Developing the Core (2018)

Rapid technological advancement took place in 2018. The launch of the Shasta Testnet in March paved the way for the pivotal Mainnet Launch, Odyssey 2.0, in May, establishing TRON as a high-performance, independent Layer-1 blockchain capable of supporting a new generation of DApps. Prioritizing network efficiency and decentralized governance, TRON implemented the Delegated Proof-of-Stake (DPoS) consensus mechanism and elected its Super Representatives (SR) in June 2018. This foundational year also saw strategic expansion with the acquisition of BitTorrent in July, significantly amplifying TRON's reach and potential within decentralized content distribution. The latter half of the year saw the launch of the Ethereum-compatible TRON Virtual Machine (TVM) in October, a key catalyst for attracting developers and fostering ecosystem growth. Supporting this development, the Odyssey-v3.2 upgrade in November introduced resource delegation, optimizing network utility. By December 2018, TRON's burgeoning ecosystem had attracted over 1 million user accounts. The introduction of the TRC-10 and TRC-20 token standards throughout 2018 provided the fundamental infrastructure for asset tokenization and DApp innovation on the platform.

[1] Whitepaper V1.0 is available at https://tron.network/static/doc/white_paper_v_1_0.pdf; whitepaper V2.0 is available at https://tron.network/static/doc/white_paper_v_2_0.pdf

1.3.3 Expanding Functionality and Adoption (2019 - 2021)

TRON continued its evolution with key protocol upgrades in 2019, including Odyssey-v3.5 in March, enhancing account security and flexibility, and Odyssey-v3.6.5 in October, optimizing staking and governance mechanisms. This period also marked the significant rise of TRON as a leading network for stablecoin transfers, with the widespread adoption of TRC-20 USDT beginning in 2019, leveraging its speed and low transaction fees. This utility became a major driver of user growth. By September 2020, the platform had onboarded over 10 million accounts. The increasing prominence of TRON in the stablecoin ecosystem was further underscored by the rapid growth of TRC-20 USDT issuance, surpassing 10 billion by early 2021 and reaching over 30 billion by May 2021.

1.3.4 Sustained Growth and Ecosystem Dominance (2022 - Present)

This period demonstrated continued robust growth in both user adoption and stablecoin dominance. TRON's user base experienced significant acceleration, scaling to over 100 million accounts by June 2022 and exceeding 300 million by April 2025. Concurrently, TRC-20 USDT issuance continued its upward trajectory, reaching over 70 billion by April 2025, solidifying TRON's position as a critical infrastructure within the global stablecoin landscape. This expansion is complemented by TRON's ongoing commitment to enhancing core platform stability, scalability, performance, and calibrating the economic model in response to market dynamics.

Significant protocol advancements during this period include a series of mandatory major upgrades. The GreatVoyage-v4.7.0.1 (Aristotle) release introduced the Stake 2.0 mechanism and a dynamic Energy model for the TVM, enhancing resource management and execution efficiency. The GreatVoyage-v4.7.2 (Periander) release then upgraded TRON's network module to Libp2p v1.2.0, improving peer-to-peer communication and block synchronization. It also introduced critical updates to Stake 2.0 and the EIP-3855 PUSH0 Instruction in the TVM. The GreatVoyage-v4.8.0 (Kant) release further enhanced the TRON Virtual Machine (TVM) by adapting to key improvements from Ethereum's recent upgrades, aiming to improve smart contract efficiency and reduce transaction costs, maintaining TRON's competitiveness as an EVM-compatible blockchain.

1.3.5 Looking Ahead

Building upon its established infrastructure, TRON remains committed to enhancing interoperability, fostering broader adoption, and further solidifying its position as a leading blockchain with a thriving ecosystem and unparalleled efficiency in stablecoin transfers within the evolving landscape of decentralized technologies.

1.4 Terminology

For definitions of key terms and acronyms used throughout this whitepaper, refer to Appendix A: Terminology.

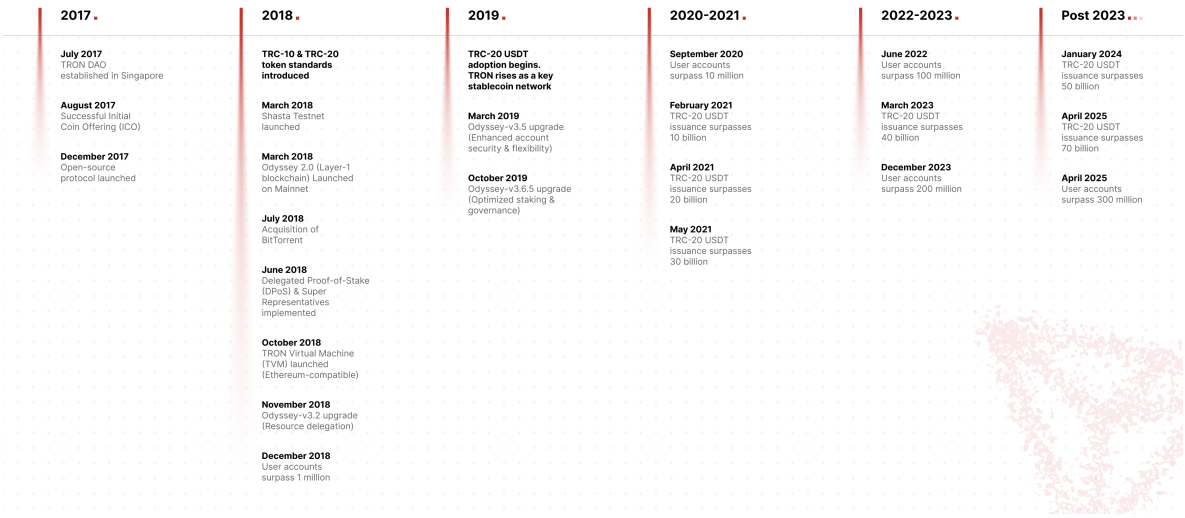

Figure 1: *TRON Development Timeline & Milestones (2017–Present)*

2 Architecture

TRON adopts a three-layer architecture composed of a Core Layer, Storage Layer, and Application Layer. The TRON protocol adheres to Google's Protocol Buffers (Protobuf), which intrinsically supports multi-language extension.

2.1 Core Layer

The Core Layer is composed of several key modules responsible for functions including smart contracts, account management, and consensus. TRON implements a stack-based virtual machine, the TRON Virtual Machine (TVM), which uses an optimized instruction set. To better support DApp developers, Solidity[2] was chosen as the smart contract language, followed by future support of other advanced languages. In addition, TRON's consensus mechanism is based on Delegated Proof of Stake (DPoS), incorporating many innovations to meet its unique requirements.

2.2 Storage Layer

TRON's unique distributed storage protocol consists of Block Storage and State Storage. The notion of a graph database was introduced into the design of the Storage Layer to better meet the need for diversified data storage in the real world.

2.2.1 Blockchain Storage

TRON blockchain storage utilizes LevelDB, which is developed by Google and proven successful with many companies and projects. It has high performance and supports arbitrary byte arrays as both keys and values, operations on individual keys (get, put, delete), batch operations (put, delete), bi-directional iterators, and simple compression using the fast Snappy algorithm.

2.2.2 State Storage

TRON has a KhaosDB in the full-node memory that can store all the newly forked chains generated within a certain period of time. This enables SRs to switch swiftly from their own active chain into a new main chain. It also protects the blockchain storage by making it more stable from being terminated abnormally in an intermediate state.

[2]Solidity official documentation: https://solidity.readthedocs.io/

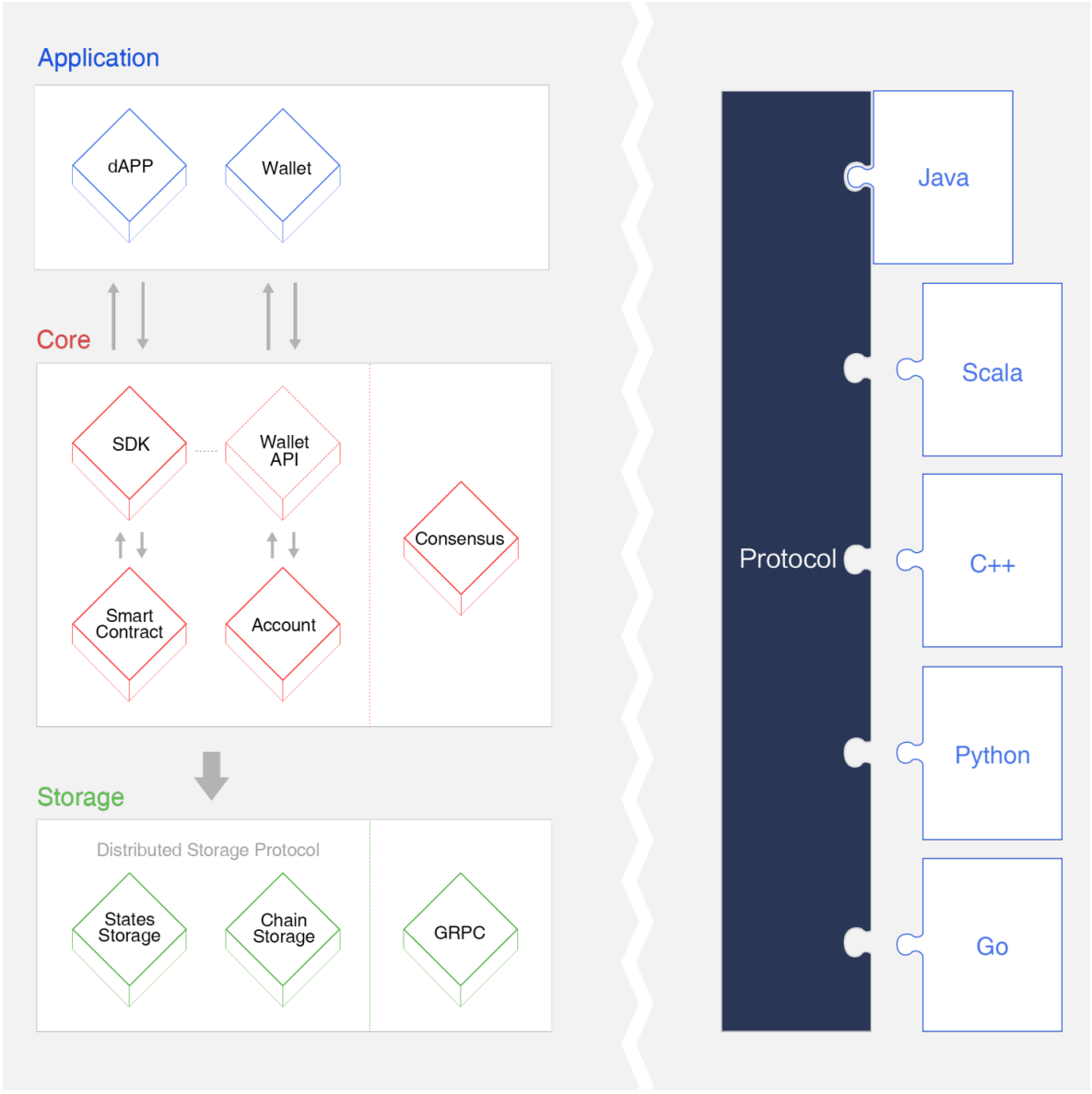

Figure 2: *TRON 3-layer Architecture*

2.3 Application Layer

Developers can create a diverse range of DApps and customized wallets on TRON. Since TRON enables smart contracts to be deployed and executed, it supports a wide spectrum of potential use cases.

2.4 Protocol

TRON protocol adheres to Google Protocol Buffers[3], which is a language-neutral, platform-neutral, and extensible way of serializing structured data for use in communications protocols, data storage, and more.

2.4.1 Protocol Buffers

Protocol Buffers (Protobuf) is a flexible, efficient, automated mechanism for serializing structured data, similar to JSON or XML, but is significantly smaller, faster, and simpler.

Protobuf (.proto) definitions can be used to generate code for C++, Java, C#, Python, Ruby, Golang, and Objective-C languages through the official code generators. Various third-party implementations are also available for many other languages. Protobuf simplifies client development by unifying the API definitions and also optimizing data transfers. Clients can use the API.proto from TRON's protocol repository and integrate through the automatically-generated code libraries.

As a comparison, Protocol Buffers is 3 to 10 times smaller and 20 to 100 times faster than XML, with less ambiguous syntax. Protobuf generates data access classes that are easier to use programmatically.

2.4.2 HTTP

TRON Protocol provides a RESTful HTTP API alternative to the Protobuf API. They share the same interface but the HTTP API can be readily used in JavaScript clients.

2.5 TRON Virtual Machine (TVM)

The TVM is a lightweight, Turing-complete virtual machine developed for TRON's ecosystem. The TVM connects seamlessly with the existing development ecosystem to provide millions of global developers with a custom-built blockchain system that is efficient, convenient, stable, secure, and scalable.

[3]Google Protocol Buffers official documentation: https://developers.google.com/protocolbuffers/

3 Consensus

3.1 Delegated Proof of Stake (DPoS)

The earliest consensus mechanism is the Proof of Work (PoW) consensus mechanism. This protocol is currently implemented in Bitcoin[4] and Ethereum[5]. In PoW systems, transactions broadcast through the network are grouped into nascent blocks for miner confirmation. This confirmation process involves hashing transactions using cryptographic hashing algorithms until a merkle root has been reached, creating a merkle tree:

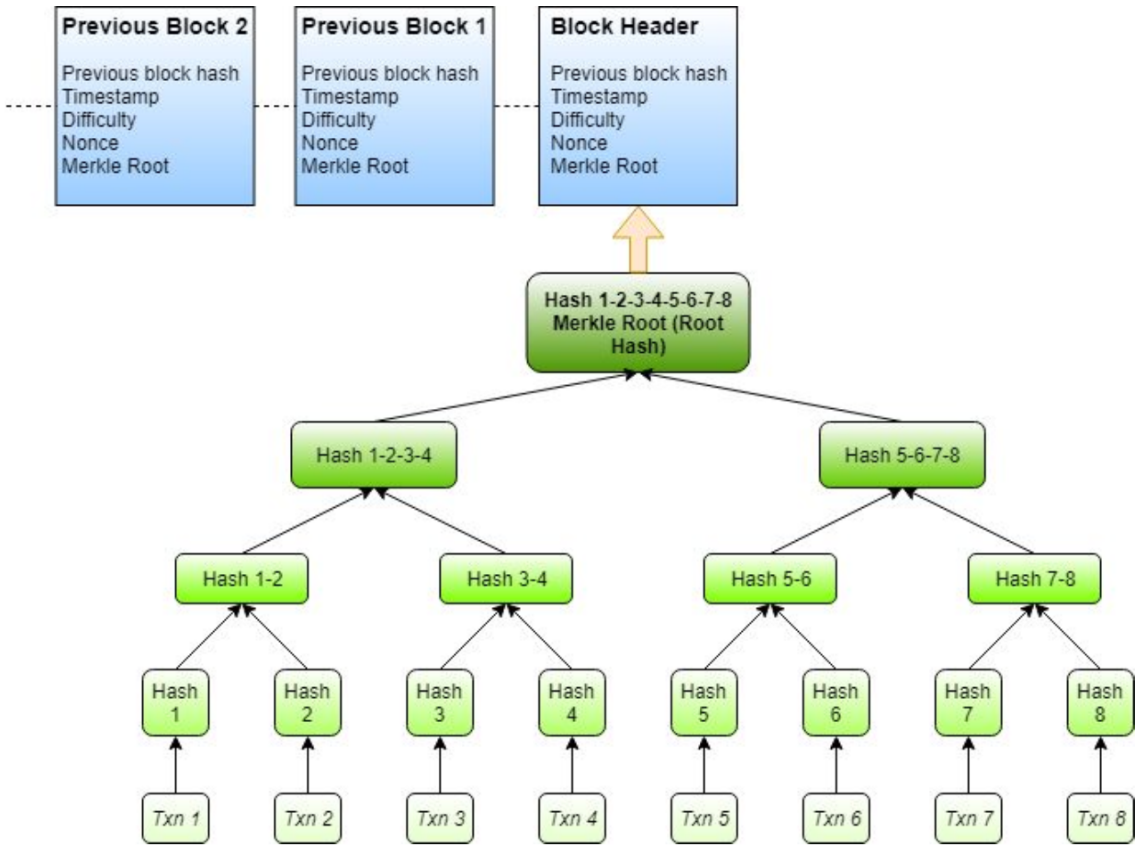

Figure 3: *8 TRX transactions are hashed into the merkle root. This merkle root is then included in the block header, which is attached to the previously confirmed blocks to form a blockchain. This allows for easy and transparent tracking of transactions, timestamps, and other related information.*

Cryptographic hashing algorithms are useful in network attack prevention because they possess several properties[6]:

- ***Input/Output length size*** - The algorithm can pass in an input of any length in size, and outputs a fixed-length hash value.
- ***Efficiency*** - The algorithm is relatively easy and fast to compute.
- ***Preimage resistance*** - For a given output z, it is impossible to find any input x such that $h(x) = z$. In other words, the hashing algorithm $h(x)$ is a one-way function in which only the output

[4]Bitcoin whitepaper: https://bitcoin.org/bitcoin.pdf
[5]Ethereum whitepaper: https://github.com/ethereum/wiki/wiki/WhitePaper
[6]PAAR, C., PELZL, J., Understanding Cryptography: A Textbook for Students and Practitioners, 2010 ed. Springer-Verlag Berlin Heidelberg, 2010.

can be found, given an input. The reverse is not possible.

- ***Collision resistance*** - It is computationally infeasible to find any pairs $x_1 \neq x_2$ such that $h(x_1) = h(x_2)$. In other words, the probability of finding two different inputs hashing to the same output is extremely low. This property also implies *second preimage resistance*.
- ***Second preimage resistance*** - Given x_1, and thus $h(x_1)$, it is computationally infeasible to find any x_2 such that $h(x_1) = h(x_2)$. While this property is similar to *collision resistance*, the property differs in that it is saying an attacker with a given x_1 will find it computationally infeasible to find any x_2 hashing to the same output.
- ***Deterministic*** - maps each input to one and only one output.
- ***Avalanche effect*** - a small change in the input results in an entirely different output.

These properties give the cryptocurrency network its intrinsic value by ensuring attacks do not compromise the network. When miners confirm a block, they are rewarded with tokens as a built-in incentive for network participation. However, as the global cryptocurrency market capitalization steadily increased, the miners became centralized and focused their computing resources on hoarding tokens as assets, rather than for network participation purposes. CPU miners gave way to GPUs, which in turn gave way to powerful ASICs.

To solve the energy waste issue, the Proof of Stake (PoS) consensus mechanism was proposed by many new networks. In PoS networks, token holders lock their token balances to become block validators. The validators take turns proposing and voting on the next block. However, the problem with standard PoS is that validator influence correlates directly to the amount of tokens locked up. This results in parties hoarding large amounts of the network's base currency wielding undue influence in the network ecosystem.

The TRON consensus mechanism uses an innovative Delegated Proof of Stake system in which 27 SRs produce blocks for the network. Every 6 hours, TRX account holders who freeze their accounts can vote for a selection of SR candidates, with the top 27 candidates deemed the SRs. Voters may choose SRs based on criteria such as projects sponsored by SRs to increase TRX adoption, and rewards distributed to voters. This allows for a more democratized and decentralized ecosystem. SRs' accounts are normal accounts, but their accumulation of votes allows them to produce blocks. With the low throughput rates of Bitcoin and Ethereum due to their PoW consensus mechanism and scalability issues, TRON's DPoS system offers an innovative mechanism resulting in 2000 Transactions Per Second (TPS) compared to Bitcoin's 3 TPS and Ethereum's 15 TPS.

The TRON protocol network generates one block every three seconds, with each block awarding 16 TRX to SRs. A total of 168,192,000 TRX will be awarded annually to the 27 SRs. Each time an SR finishes block production, rewards are sent to a sub-account in the super-ledger. SRs can check, but not directly make use of these TRX tokens. A withdrawal can be made by each SR once every 24 hours, transferring the rewards from the sub-account to the specified SR account.

The three types of nodes on the TRON network are Witness Node, Full Node, and Lite Full Node. Witness nodes are set up by SRs and are mainly responsible for block production and proposal creation/voting. Full nodes provide APIs and broadcast transactions and blocks. Solidity nodes sync blocks from other Full Nodes and also provide indexable APIs.

4 Account

4.1 Types

The TRON network features two primary types of accounts:

1. **Externally Owned Accounts (EOAs)**: These accounts are controlled by a private key. They are used to send TRX and tokens, vote for Super Representatives, and deploy smart contracts. An EOA account can hold balances of TRX, TRC-10 tokens, and TRC-20/TRC-721 tokens (via interaction with contract accounts). This is the most common type of account, used by all wallet users.

2. **Contract Accounts**: These are smart contract accounts controlled by the logic of their code. A contract account does not have a private key and is activated upon creation. Its functions can be invoked by any EOA or other contract, as permitted by its code.

4.2 Creation

There are three ways to create a TRON account:

1. Create an account offline using the command-line wallet `wallet-cli`;

2. Create an account offline using an SDK, such as the Trident SDK;

3. Create a private key and its corresponding address via a wallet application.

Accounts can be activated in the following two ways:

1. Send any amount of TRX or TRC-10 tokens from an existing account to the new account;

2. Call Java-tron's `wallet/createaccount` API to create a transaction from an existing account, then sign the transaction and broadcast it to the TRON network.

An offline key pair consisting of an address (public key) and a private key, and not recorded by the TRON network, can also be generated. The user address generation algorithm consists of generating a key pair and then extracting the public key (64-byte byte array representing x, y coordinates). Hash the public key using the SHA3-256 function (the SHA3 protocol adopted is KECCAK-256) and extract the last 20 bytes of the result. Add 41 to the beginning of the byte array and ensure the initial address length is 21 bytes. Hash the address twice using the SHA3-256 function and take the first 4 bytes as verification code. Add the verification code to the end of the initial address and obtain the address in base58check format through base58 encoding. An encoded Mainnet address begins with T and is 34 bytes in length.

4.3 Structure

The three different account types are Normal, AssetIssue, and Contract. An Account contains 7 parameters:

- **account_name**: The name for the account – for example, BillsAccount;

- **type**: The type of the account – for example, 0 (stands for type 'Normal');

- **address**: The unique identifier for the account on the TRON network - for example, T9yD. . . 9mGg;

- **balance**: The TRX balance of the account – for example, 4213312;

- **vote**: Votes received by the account – for example, {("0x1b7w. . . 9xj3",323), ("0x8djq. . . j12m",88), . . . ,("0x82nd. . . mx6i",10001)};

- **asset**: Other assets, besides TRX, held in the account – for example, {<"WishToken", 66666>,

<”Dogie”, 233>}.

- **latest_operation_time**: The timestamp of the account's most recent operation.

Protobuf data structure:

```
message Account {
    message Vote {
        bytes vote_address = 1;
        int64 vote_count = 2;
    }
    bytes accout_name = 1;
    AccountType type = 2;
    bytes address = 3;
    int64 balance = 4;
    repeated Vote votes = 5;
    map<string, int64> asset = 6;
    int64 latest_operation_time = 10;
}
enum AccountType {
    Normal = 0;
    AssetIssue = 1;
    Contract = 2;
}
```

5 Block

A block typically contains a block header and several transactions. Protobuf data structure:

```
message Block {
    BlockHeader block_header = 1;
    repeated Transaction transactions = 2;
}
```

5.1 Block Header

A block header contains **raw_data**, **witness_signature**, and **blockID**. Protobuf data structure:

```
message BlockHeader {
    message raw {
        int64 timestamp = 1;
        bytes txTrieRoot = 2;
        bytes parentHash = 3;
        int64 number = 7;
        int64 witness_id = 8;
        bytes witness_address = 9;
        int32 version = 10;
        bytes accountStateRoot = 11;
    }
    raw raw_data = 1;
    bytes witness_signature = 2;
}
```

5.1.1 Raw Data

Raw data is denoted as **raw_data** in Protobuf. It contains the raw data of a message, containing 8 parameters:

1. **timestamp**: timestamp of this message – for example, 1543884429000.
2. **txTrieRoot**: the Merkle Tree's Root – for example, 7dacsa...3ed.
3. **parentHash**: the hash of the last block – for example, 7dacsa...3ed.
4. **number**: the block height – for example, 4638708.
5. **version**: reserved – for example, 5.
6. **witness_address**: the address of the witness packed in this block – for example, 41928c...4d21.
7. **witness_id**: the ID of the witness that packed this block – for example, "_0xu82h...7237_".
8. **accountStateRoot**: the address of the witness packed this block – for example, "_0xu82h...7237_".

5.1.2 Witness Signature

Witness signature is denoted as **witness_signature** in Protobuf, which is the signature for this block header from the witness node.

5.1.3 Block ID

Block ID is denoted as **blockID** in Protobuf. It contains the atomic identification of a block. A Block ID contains 2 parameters:

1. **hash**: the hash of block.
2. **number**: the hash and height of the block.

5.2 Transaction

5.2.1 Signing

TRON's transaction signing process follows a standard ECDSA cryptographic algorithm, with a SECP256K1 selection curve. A private key is a random number, and the public key is a point on the elliptic curve. The public key generation process consists of first generating a random number as a private key, and then multiplying the base point of the elliptic curve by the private key to obtain the public key. When a transaction occurs, the transaction raw data is first converted into byte format.

The raw data then undergoes SHA-256 hashing. The private key corresponding to the contract address then signs the result of the SHA256 hash. The signature result is then added to the transaction.

5.2.2 Bandwidth Model

Ordinary transactions only consume Bandwidth Points, but smart contract operations consume both Energy and Bandwidth Points. There are two types of Bandwidth Points available. Users can gain Bandwidth Points from freezing TRX, while free daily Bandwidth Points are also available daily[7].

When a TRX transaction is broadcasted, it is transmitted and stored in the form of a byte array over the network. Bandwidth Points consumed by one transaction = number of transaction bytes multiplied by Bandwidth Points rate. For example, if the byte array length of a transaction is 200, then the transaction consumes 200 Bandwidth Points. However, if a TRX or token transfer results in the target account being created, then only the Bandwidth Points consumed to create the account will be deducted, and additional Bandwidth Points will not be deducted. When a transfer creates an account lacking sufficient available Bandwidth, a proposal-controlled account creation fee of 0.1 TRX will be deducted. Account creation transactions currently have a size limit of 1000 bytes, also governed by proposal. In an account creation scenario, the network will first consume the Bandwidth Points that the transaction initiator gained from freezing TRX. If this amount is insufficient, then the network consumes the transaction initiator's TRX.

In standard TRX transfer scenarios from one TRX account to another, the network first consumes the Bandwidth Points gained by the transaction initiator for freezing TRX. If that is insufficient, it then consumes from the daily free Bandwidth Points. If that is still not enough, then the network consumes the TRX of the transaction initiator. The amount is calculated by the number of bytes in the transaction multiplied by the TRX cost per byte[8]. Thus, for most TRX holders who may not necessarily freeze their TRX to participate in SR voting, the first step is automatically skipped (since TRX balance frozen = 0) and the daily free Bandwidth powers the transaction.

For TRC-10 token transfers, the network first verifies whether the total free Bandwidth Points of the issued token asset are sufficient. If not, the Bandwidth Points obtained from freezing TRX are consumed. If there are still not enough Bandwidth Points, then it consumes the TRX of the transaction initiator.

5.2.3 Fee

TRON network generally does not charge fees for most transactions, however, due to system restrictions and fairness, Bandwidth usage and transactions do incur certain fees.

Fee charges are broken down into the following categories:

1. Normal transactions cost Bandwidth Points. Users can use the free daily Bandwidth Points or freeze TRX to obtain more. When Bandwidth Points are not enough, TRX will be used directly from the sending account. The TRX needed is the number of bytes * TRX cost per byte.

2. Smart contracts cost Energy (Section 6) but will also need Bandwidth Points for the transaction to be broadcasted and confirmed. The Bandwidth cost is the same as above.

[7] The specific amount of this free allowance is defined by the `freeNetLimit` parameter in the TRON Chain Parameters API: https://developers.tron.network/reference/wallet-getchainparameters

[8] This cost is defined by the getTransactionFee parameter in the TRON Chain Parameters API

3. All query transactions are free. It doesn't cost Energy or Bandwidth.

TRON network also defines a set of fixed fees for the following transactions (for the latest values, please refer to the TRON Chain Parameters API):

1. Issue a TRC-10 token: 1,024 TRX (see `getAssetIssueFee`)
2. Apply to be an SR candidate: 9,999 TRX (see `getAccountUpgradeCost`)
3. Create a Bancor transaction: 1,024 TRX (see `getExchangeCreateFee`)
4. Update the account permission: 100 TRX (see `getUpdateAccountPermissionFee`)
5. Activate the account: 1 TRX (see `getCreateNewAccountFeeInSystemContract`); insufficient Bandwidth obtained through staking in the creator's account incurs an additional 0.1 TRX fee to pay for the Bandwidth (see `getCreateAccountFee`)
6. Multisig transaction: 1 TRX (see `getMultiSignFee`)
7. Transaction note: 1 TRX (see `getMemoFee`)

5.2.4 Transaction as Proof of Stake (TaPoS)

TRON uses TaPoS to ensure the transactions all confirm the main blockchain, while making it difficult to forge counterfeit chains. In TaPoS, the networks require each transaction include part of the hash of a recent block header. This requirement prevents transactions from being replayed on forks not including the referenced block, and also signals the network that a particular user and their stake are on a specific fork. This consensus mechanism protects the network against Denial of Service, 51%, selfish mining, and double-spending attacks.

5.2.5 Transaction Confirmation

A transaction is included in a block after being broadcast to the network. The transaction is considered confirmed after its block has been followed by 19 subsequent blocks, each produced by a different Super Representative.

Each block takes $\tilde{3}$ seconds to be mined on the blockchain. Time may slightly vary for each SR due to network conditions and machine configurations. In general, a transaction is considered fully confirmed after $\tilde{1}$ minute.

5.2.6 Structure

Transaction APIs consist of the following functions:

```
message Transaction {
    message Contract {
        enum ContractType {
            AccountCreateContract = 0; // Create account/wallet
            TransferContract = 1; // Transfer TRX
            TransferAssetContract = 2; // Transfer TRC-10 token
            VoteWitnessContract = 4; // Vote for SR
            WitnessCreateContract = 5; // Create a new SR account
            AssetIssueContract = 6; // Create a new TRC-10 token
            WitnessUpdateContract = 8; // Update SR information
            ParticipateAssetIssueContract = 9; // Purchase TRC-10 token
            AccountUpdateContract = 10; // Update account/wallet information
            FreezeBalanceContract = 11; // Freeze TRX for Bandwidth or Energy
            UnfreezeBalanceContract = 12; // Unfreeze TRX
            WithdrawBalanceContract = 13; // Withdraw SR rewards, once per day
            UnfreezeAssetContract = 14; // Unfreeze TRC-10 token
            UpdateAssetContract = 15; // Update a TRC-10 token's information
            ProposalCreateContract = 16; // Create a new network proposal by any SR
            ProposalApproveContract = 17; // SR votes yes for a network proposal
            ProposalDeleteContract = 18; // Delete a network proposal by owner
```

```
            CreateSmartContract = 30; // Deploy a new smart contract
            TriggerSmartContract = 31; // Call a function on a smart contract
            GetContract = 32; // Get an existing smart contract
            UpdateSettingContract = 33; // Update a smart contract's parameters
            ExchangeCreateContract = 41; // Create a token trading pair on DEX
            ExchangeInjectContract = 42; // Inject funding into a trading pair
            ExchangeWithdrawContract = 43; // Withdraw funding from a trading pair
            ExchangeTransactionContract = 44; // Perform token trading
            UpdateEnergyLimitContract = 45; // Update origin_energy_limit on a smart
    contract
            AccountPermissionUpdateContract = 46; // Update account permissions
            ClearABIContract = 48; // Clear a smart contract's ABI
            UpdateBrokerageContract = 49; // Update SR brokerage commission rate
            MarketSellAssetContract = 52; // Place a market sell order
            MarketCancelOrderContract = 53; // Cancel an existing market order
            FreezeBalanceV2Contract = 54; // Freeze TRX to get resources
            UnfreezeBalanceV2Contract = 55; // Cancel freezed TRX
            WithdrawExpireUnfreezeContract = 56; // Withdraw TRX after the unfreeze waiting
    period has expired
            DelegateResourceContract = 57; // Delegate Bandwidth or Energy to another
    account
            UnDelegateResourceContract = 58; // Cancel delegated resource
            CancelAllUnfreezeV2Contract = 59; // Cancel all pending unfreeze operations
        }
    ContractType type = 1;
    google.protobuf.Any parameter = 2;
    bytes provider = 3;
    bytes ContractName = 4;
    int32 Permission_id = 5;
    }
}
```

6 TRON Virtual Machine (TVM)

6.1 Introduction

The TRON Virtual Machine (TVM) is a lightweight, Turing-complete virtual machine developed for the TRON ecosystem. Its goal is to provide a custom-built blockchain system that is efficient, convenient, stable, secure, and scalable.

The TVM initially forked from the EVM[9] and can connect seamlessly with the existing Solidity smart contract development ecosystem. Based on that, the TVM additionally supports DPoS consensus.

The TVM introduces the concept of Energy to manage computational resources, which differs from the EVM's Gas mechanism. On the TVM, the execution of smart contracts consumes Energy. Users can obtain Energy by staking TRX. When an account has sufficient Energy, executing smart contract operations does not require the direct burning of TRX. When available Energy is insufficient, the system will then burn TRX from the account to cover the computational costs.

6.2 Workflow

The compiler first translates the Solidity smart contract into bytecode readable and executable on the TVM. The TVM then processes data through opcode, which is equivalent to operating the logic of a stack-based finite state machine. Finally, the TVM accesses blockchain data and invokes the External Data Interface through the Interoperation layer.

6.3 Performance

6.3.1 Lightweight Architecture

The TVM adopts a lightweight architecture with the aim of reducing resource consumption to guarantee system performance.

6.3.2 Robust

TRON employs a dual-resource model (comprising Bandwidth and Energy) for transaction processing and smart contract execution. Users can primarily acquire these resources without direct transactional cost by freezing TRX. When resource limits are exceeded, TRX is consumed (burned) proportionally to the excess usage. This resource management system enhances network security by significantly increasing the economic cost of malicious activities and ensures predictable resource consumption due to fixed unit costs for each operation.

6.3.3 High Compatibility

The TVM is compatible with the EVM and will be compatible with more mainstream VMs in the future. Thereby, all smart contracts on the EVM are executable on the TVM.

6.3.4 Low Cost

Due to the TVM's Bandwidth setup, development costs are reduced, and developers can focus on the logic development of their contract code. The TVM also offers all-in-one interfaces for contract deployment, triggering, and viewing to offer the convenience for developers.

[9] EVM: Ethereum Virtual Machine (https://github.com/ethereum/ethereumj)

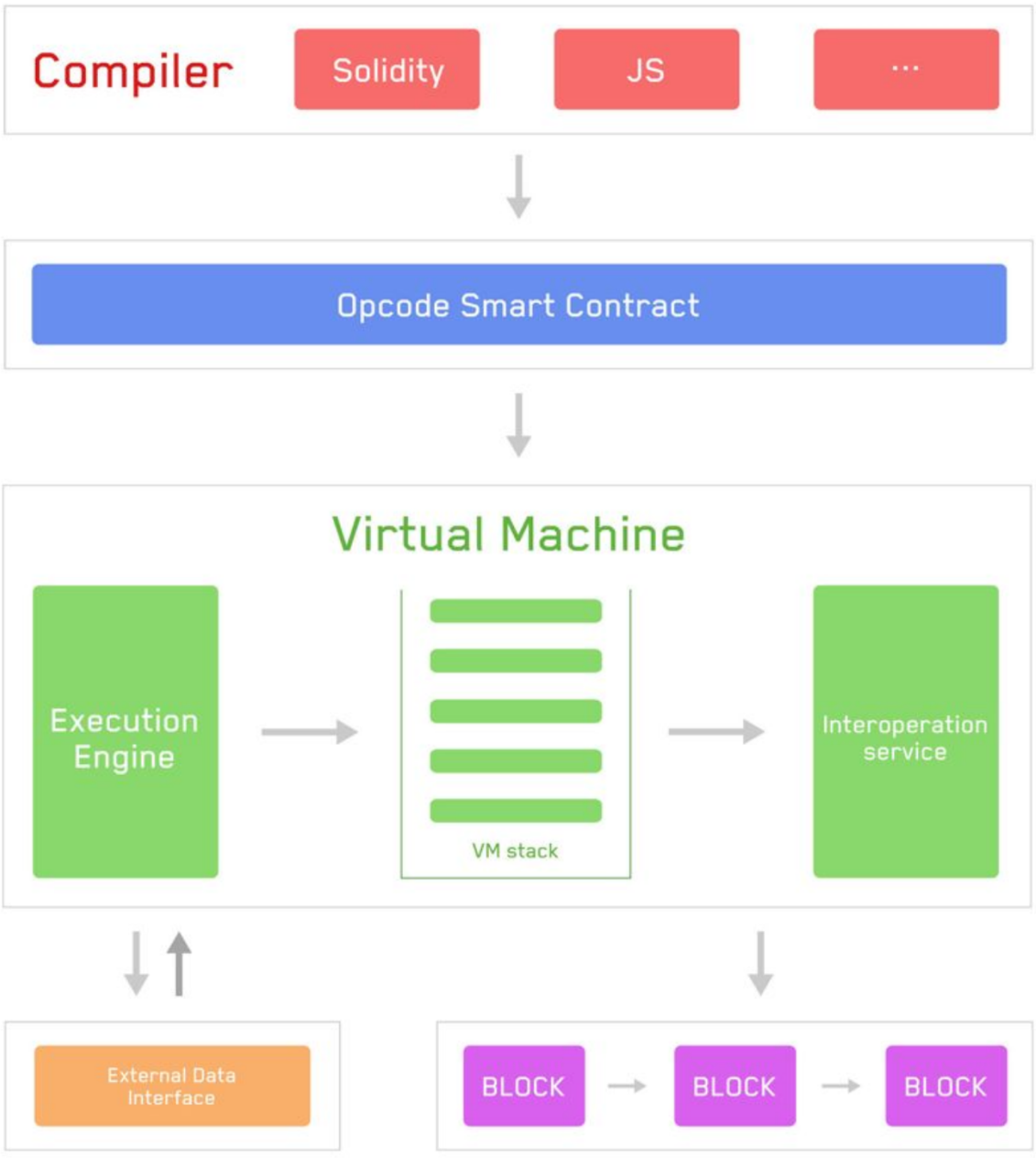

Figure 4: *TVM Workflow*

7 Smart Contract

7.1 Introduction

A smart contract is a protocol that digitally verifies contract negotiation. They define the rules and penalties related to an agreement and also automatically enforce those obligations. The smart contract code facilitates, verifies, and enforces the negotiation or performance of an agreement or transaction. From a tokenization perspective, smart contracts also facilitate automatic funds transfers between participating parties should certain criteria be met.

TRON smart contracts are written in the Solidity language. Once written and tested, they can be compiled into bytecode, then deployed onto the TRON network for the TRON Virtual Machine. Once deployed, smart contracts can be queried via their contract addresses. The contract Application Binary Interface (ABI) shows the contract's call functions and is used for interacting with the network.

7.2 Energy Model

The maximum Energy limit for deploying and triggering a smart contract is a function of several variables:

- Dynamic Energy from freezing 1 TRX is 180,000,000,000 (Total Energy Limit) / (Total Energy Weight)
- Energy limit is the daily account Energy limit from freezing TRX
- Remaining daily account Energy from freezing TRX is calculated as Energy Limit - Energy Used
- Fee limit in TRX is set in smart contract deploy/trigger call
- Remaining usable TRX in the account
- Energy per TRX if purchased directly (210 SUN = 1 Energy) = 100,000, SRs can vote on adjustment

There are two consumption scenarios to calculate for maximum Energy limit for deployment and trigger. The logic can be expressed as follows:

```
    const R = Dynamic Energy Limit
    const F = Daily account Energy from freezing TRX
    const E = Remaining daily account Energy from freezing TRX
    const L = Fee limit in TRX set in deploy/trigger call
    const T = Remaining usable TRX in account
    const C = Energy per TRX if purchased directly
    // Calculate M, defined as maximum Energy limit for deployment/trigger of a smart
    contract
    if F > L*R
        let M = min(E+T*C, L*R)
    else
        let M = E+T*C
```

Since February 5, 2023, the TRON network has implemented a new dynamic Energy model, enabled by the No. 83 committee proposal initiated by the TRON developer community. This model dynamically adjusts the Energy consumption of each contract according to the contract's resource occupancy. This aims to make the allocation of Energy resources on the chain more reasonable and to prevent excessive concentration of network resources on a few popular contracts.

7.3 Deployment

When a TRON solidity smart contract is compiled, the TRON Virtual Machine reads the compiled bytecode. The bytecode consists of a section for code deployment, contract code, and the Auxdata. The Auxdata is the source code's cryptographic fingerprint, used for verification. The deployment bytecode runs the constructor function and sets up the initial storage variables. The deployment code

also calculates the contract code and returns it to the TVM. The ABI is a JSON file that describes a TRON smart contract's functions. This file defines the function names, their payability, the function return values, and their state mutability.

7.4 Trigger Function

Once the TRON smart contracts are deployed, their functions can be triggered individually either via Tronide or through API calls. State-changing functions require Energy while read-only functions execute without Energy.

7.5 TRON Solidity

TRON Solidity is a fork from Ethereum's Solidity language. TRON modifies the original project to support TRX and SUN units (1 TRX = 1,000,000 SUN). The rest of the language syntax is compatible with Solidity 0.8.23. Thus the Tron Virtual Machine (TVM) is almost 100% compatible with EVM instructions.

8 Token

8.1 TRC-10 Token

In the TRON network, each account can issue tokens at the expense of 1024 TRX. To issue tokens, the issuer needs to specify a token name, the total capitalization, the exchange rate to TRX, circulation duration, description, website, maximum Bandwidth consumption per account, total Bandwidth consumption, and the amount of tokens frozen. Each token issuance can also configure each account's maximum daily token transfer Bandwidth Points, the entire network's maximum daily token transfer Bandwidth Points, total token supply, locking duration in days, and the total amount of tokens locked.

8.2 TRC-20 Token

TRC-20 is a technical standard used for smart contracts implementing tokens supported by the TRON Virtual Machine. It is fully compatible with ERC-20.

The interface is as follows:

```
contract TRC20Interface {
    function totalSupply() public constant returns (uint);
    function balanceOf(address tokenOwner) public constant returns (uint balance);
    function allowance(address tokenOwner, address spender) public constant returns (uint
    remaining);
    function transfer(address to, uint tokens) public returns (bool success); function
    approve(address spender, uint tokens) public returns (bool success);
    function transferFrom(address from, address to, uint tokens) public returns (bool
    success);
    event Transfer(address indexed from, address indexed to, uint tokens); event Approval(
    address indexed tokenOwner, address indexed spender, uint tokens);
}
```

Transfers of TRC-10 tokens primarily consume Bandwidth, whereas TRC-20 token transfers consume both Bandwidth and Energy. This difference in resource consumption often results in significantly lower transaction fees for TRC-10 tokens compared to TRC-20. While API transfers and deposits of TRC-10 tokens incur Bandwidth costs, the cost remains generally lower than the combined Bandwidth and Energy of TRC-20 transactions. Transfers and deposits in smart contracts for TRC-10 tokens cost both Bandwidth and Energy. In contrast, TRC-20 consistently consumes both Bandwidth and Energy for transfers and deposits, whether through API calls or smart contracts.

8.3 Beyond

Since TRON uses the same Solidity version as Ethereum, more token standards could be readily ported to TRON.

9 Governance

9.1 Super Representative

9.1.1 General

Every account in the TRON network can apply and have the opportunity to become a Super Representative (SR). Everyone can vote for SR candidates. The top 27 candidates with the most votes will become SRs with the right and obligation to generate blocks. The votes are counted every 6 hours and the SRs will change accordingly.

To prevent malicious attacks, there is a cost to becoming an SR candidate. When applying, 9999 TRX will be burned from the applicant's account. Once successful, such account can join the SR election.

9.1.2 Election

TRON Power (denoted as TP) is needed to vote and the amount of TP depends on the voter's frozen assets (TRX).

TP is calculated in the following way:

$$1\,TP\ = 1\,TRX\ frozen\ to\ get\ Energy/Bandwidth$$

Every account in the TRON network has the right to vote for their own SRs.

After the release (unfreeze, available after 14 days), users won't have any frozen assets and lose all TP accordingly. As a result, all votes become invalid for the ongoing and future voting round unless TRX is frozen again to vote.

Note that the TRON network only records the most recent vote, which means that every new vote will negate all previous votes.

9.1.3 Reward

a. Vote Reward

Also known as Candidate Reward, which the top 127 candidates updated once every round (6 hours) will share 921,600 TRX as mined. The reward will be split in accordance with the vote weight each candidate receives. Each year, the total reward for candidates will be 1,345,536,000 TRX.

Total vote reward per round

Why 921,600 TRX every round?

$$921,600\ \ TRX\ = total\ vote\ reward\ per\ round\ (V\ R/round)$$

$$V\ R/round\ = 128\ \ TRX/block\ \times 20\ \ blocks/min\ \times 60\ \ mins/hr\ \times 6\ \ hrs/round$$

Notice: this is set by `WITNESS_127_PAY_PER_BLOCK` = 128 TRX. See dynamic network parameters.

Total vote reward per year

Why 1,345,536,000 TRX every year?

$$1,345,536,000\ \ TRX\ = total\ vote\ reward\ per\ year\ (V\ R/year)$$

$$V\ R/year\ = 921,600\ \ TRX/round\ \times 4\ \ rounds/day\ \times 365\ \ days/year$$

b. Block Reward

Also known as Super Representative Reward, which the top 27 candidates (SRs) who are elected every round (6 hours) will share roughly 57,600 TRX as mined. The reward will be split evenly between the 27 SRs (minus the total reward blocks missed due to network error). A total of 84,096,000 TRX will be awarded annually to the 27 SRs.

Total block reward per round

Why 57,600 TRX every round?

$$57,600\ TRX\ = total\ block\ reward\ per\ round\ (BR/round)$$

$$BR/round\ = 8\ TRX/bloc\ \times 20\ blocks/min\ \times 60\ mins/hr\ \times 6hrs/round$$

Notice: the unit block reward is set by `WITNESS_PAY_PER_BLOCK` = 8 TRX. See dynamic network parameters.

Total block reward per year

Why 84,096,000 TRX every year?

$$84,096,000\ TRX\ = total\ block\ reward\ per\ year\ (BR/year)$$

$$BR/year\ = 57,600\ TRX/round\ \times 4\ rounds/day\ \times 365\ days/year$$

c. Reward Calculation

SR reward calculation

$$\text{total reward} = \text{vote reward}(VR) + \text{block reward}(BR)$$

$$VR = \text{total } VR \times \frac{\text{votes SR candidate received}}{\text{total votes}}$$

$$BR = \frac{\text{total } BR}{27} - \text{block missed} \times 32$$

Note: the reward is calculated per SR per round (6 hours)

Rank 28 to rank 127 SR candidate reward calculation

$$\text{total reward} = \text{vote reward}(VR)$$

$$VR = \text{total } VR \times \frac{\text{votes SR candidate received}}{\text{total votes}}$$

Note: the reward is calculated per SR candidate per round (6 hours)

9.2 Committee

9.2.1 General

The committee is used to modify TRON dynamic network parameters, such as block generation rewards, transaction fees, etc. The committee consists of the 27 SRs in the current round. Each SR has the right to propose and vote on proposals. When a proposal receives 19 votes or more, it is approved and the new network parameters will be applied in the next maintenance period (3 days).

9.2.2 Dynamic Network Parameters

Please see Appendix B for details.

9.2.3 Create Proposal

Only the SR accounts have the rights to propose a change in dynamic network parameters.

9.2.4 Vote Proposal

Only committee members (SRs) can vote for a proposal and the member who does not vote in time will be considered as a disagree. The proposal is active for 3 days after it is created. The vote can be changed or retrieved during the 3-days voting window. Once the period ends, the proposal will either succeed (19+ votes) or fail (and end).

9.2.5 Cancel Proposal

The proposer can cancel the proposal before it becomes effective.

9.3 Structure

SRs are the witnesses of newly generated blocks. A witness contains 8 parameters:

1. **address**: the address of this witness – for example, 0xu82h. . . 7237.
2. **voteCount**: number of received votes on this witness – for example, 234234.
3. **pubKey**: the public key for this witness – for example, 0xu82h. . . 7237.
4. **url**: the url for this witness – for example, https://www.noonetrust.com.
5. **totalProduced**: the number of blocks this witness produced – for example, 2434.
6. **totalMissed**: the number of blocks this witness missed – for example, 7.
7. **latestBlockNum**: the latest height of block – for example, 4522.
8. **isjobs**: a boolean flag.

Protobuf data structure:

```
message Witness{
    bytes address = 1;
    int64 voteCount = 2;
    bytes pubKey = 3;
    string url = 4;
    int64 totalProduced = 5;
    int64 totalMissed = 6;
    int64 latestBlockNum = 7;
    bool isJobs = 8;
}
```

10 DApp Development

10.1 APIs

The TRON network offers a wide selection of over 60+ HTTP API gateways for interacting with the network via Full Nodes. Additionally, TronWeb is a comprehensive JavaScript library containing API functions that enable developers to deploy smart contracts, change the blockchain state, query blockchain and contract information, trade on the DEX, and much more. These API gateways can be directed towards a local privatenet, the Shasta testnet, the Nile testnet, or the TRON Mainnet.

10.2 Networks

TRON has a Shasta testnet, a Nile testnet, as well as a Mainnet. Developers may connect to the networks by deploying nodes, interacting via Fullnode APIs the TronGrid service. The TronGrid service consists of load balanced node clusters hosted on AWS servers worldwide. As DApp development scales up and API call volumes increase, TronGrid successfully fields the increase in API traffic.

10.3 Resources

The TRON Developer Hub is a comprehensive API documentation[10] site tailored towards developers wishing to build on the TRON network. The Developer Hub provides a high-level conceptual understanding of TRON and walks users through the details of interacting with the network. The guides walk developers through node setup, deployment and interaction with smart contracts, API interaction and implementation, building sample DApps, and using each of the developer tools.

[10]TRON Developer Hub: https://developers.tron.network/

11 Conclusion

TRON is a scalable blockchain solution that has employed innovative methods for tackling challenges faced by legacy blockchain networks. Having reached over 2M transactions per day, with over 700K TRX accounts, and surpassing 2000 TPS, TRON has enabled the community in creating a decentralized and democratized network.

Appendix A: Terminology

Address/Wallet

An address or wallet consisting of account credentials on the TRON network are generated by a key pair, which consists of a private key and a public key, the latter being derived from the former through an algorithm. The public key is usually used for session key encryption, signature verification, and encrypting data that could be decrypted by a corresponding private key.

Application Binary Interface (ABI)

ABI is an interface between two binary program modules; usually one of these modules is a library or an operating system facility, and the other is a user run program.

Application Programming Interface (API)

An API is mainly used for user clients development. With API support, token issuance platforms can also be designed by developers themselves.

Asset

In TRON's documents, asset is the same as token, which is also denoted as TRC-10 token.

Bandwidth Points (BP)

To keep the network operating smoothly, TRON network transactions use BP as fuel. Each account gets a daily free amount of BP, defined by the `freeNetLimit` parameter in the TRON Chain Parameters API, and more can be obtained by freezing TRX for BP. Both TRX and TRC-10 token transfers are normal transactions costing BP. Smart contract deployment and execution transactions consume both BP and Energy.

Block

Blocks contain the digital records of transactions. A complete block consists of the magic number, block size, block header, transaction counter, and transaction data.

Block Reward

Block production rewards are sent to a sub-account (address/wallet). SRs can claim their rewards on Tronscan or through the API directly.

Block Header

A block header is part of a block. TRON block headers contain the previous block's hash, the Merkle root, timestamp, version, and witness address.

Cold Wallet

Cold wallet, also known as offline wallet, keeps the private key completely disconnected from any network. Cold wallets are usually installed on "cold" devices (for example, computers or mobile phones staying offline) to ensure the security of TRX private key.

Decentralized Application (DApp)

DApp is an application that operates without a centrally trusted party. An application that enables direct interaction/agreements/communication between end users and/or resources without a middle-man.

gRPC Remote Procedure Calls (gRPC)

gRPC[11] is an open-source remote procedure call (RPC) system initially developed at Google. It uses HTTP/2 for transport, Protocol Buffers as the interface description language, and provides features such as authentication, bidirectional streaming and flow control, blocking or nonblocking bindings, and cancellation and timeouts. It generates cross-platform client and server bindings for many languages. Most common usage scenarios include connecting services in microservices style architecture and connecting mobile devices, and browser clients to backend services.

Hot Wallet

Hot wallet, also known as online wallet, allows users' private keys to be used online, thus it could be susceptible to potential vulnerabilities or interception by malicious actors.

Java Development Kit (JDK)

JDK is the Java SDK used for Java applications. It is the core of Java development, comprising the Java application environment (JVM+Java class library) and Java tools.

KhaosDB

TRON has a KhaosDB in the full-node memory that can store all the newly-forked chains generated within a certain period of time and supports witnesses to switch from their own active chain swiftly into a new main chain. See 2.2.2 State Storage for more details.

LevelDB

LevelDB was initially adopted with the primary goal to meet the requirements of fast R/W and rapid development. After launching the Mainnet, TRON upgraded its database to an entirely customized one catered to its very own needs. See 2.2.1 Blockchain Storage for more details.

Merkle Root

A Merkle root is the hash of all hashes of all transactions included as part of a block in a blockchain network. See 3.1 Delegated Proof of Stake (DPoS) for more details.

Public Testnet (Shasta, Nile)

A version of the network running in a single-node configuration. Developers can connect and test features without worrying about the economic loss. Testnet tokens have no value and anyone can request more from the public faucet.

remote procedure call (RPC)

In distributed computing, an RPC[12] is when a computer program causes a procedure (subroutine) to execute in a different address space (commonly on another computer on a shared network), which is coded as if it were a normal (local) procedure call, without the programmer explicitly coding the details for the remote interaction.

Scalability

Scalability is a feature of the TRON Protocol. It is the capability of a system, network, or process to handle a growing amount of work or its potential to be enlarged to accommodate that growth.

[11] https://en.wikipedia.org/wiki/GRPC
[12] https://en.wikipedia.org/wiki/Remote_procedure_call

SUN

SUN replaced drop as the smallest unit of TRX. 1 TRX = 1,000,000 SUN.

Throughput

High throughput is a feature of TRON Mainnet. It is measured in Transactions Per Second (TPS), namely the maximum transaction capacity in one second.

Timestamp

The approximate time of block production is recorded as Unix timestamp, which is the number of milliseconds that have elapsed since 00:00:00 01 Jan 1970 UTC.

TRC-10

A standard of crypto token on the TRON platform. Certain rules and interfaces are required to follow when holding an initial coin offering on the TRON blockchain.

TRX

TRX stands for Tronix, which is the official cryptocurrency of TRON.

Appendix B: Dynamic Network Parameters

0. MAINTENANCE_TIME_INTERVAL

- Description: Modify the maintenance interval time, also the interval for SRs to calculate and distribute voting rewards.
- Example: `[6 * 3600 * 1000] ms` - which is 6 hours.
- Range: `[3 * 27 * 1000, 24 * 3600 * 1000] ms`

1. ACCOUNT_UPGRADE_COST

- Description: Modify the cost for an account to apply to become a Super Representative candidate.
- Example: `[9_999_000_000] SUN` - which is 9999 TRX.
- Range: `[0, 100_000_000_000_000_000] SUN`

2. CREATE_ACCOUNT_FEE

- Description: Modify the fee for creating a new account by sending TRX to a new address.
- Example: `[100_000] SUN` - which is 0.1 TRX.
- Range: `[0, 100_000_000_000_000_000] SUN`

3. TRANSACTION_FEE

- Description: Modify the fee rate for consuming Bandwidth Points when an account's free/staked Bandwidth is insufficient.
- Example: `[10] SUN/byte`.
- Range: `[0, 100_000_000_000] SUN/byte`

4. ASSET_ISSUE_FEE

- Description: Modify the one-time fee for issuing a new TRC-10 token.
- Example: `[1_024_000_000] SUN` - which is 1024 TRX.
- Range: `[0, 100_000_000_000_000_000] SUN`

5. WITNESS_PAY_PER_BLOCK

- Description: Modify the reward paid to a Super Representative for successfully producing a block.
- Example: `[16_000_000] SUN` - which is 16 TRX.
- Range: `[0, 100_000_000_000_000_000] SUN`

6. WITNESS_STANDBY_ALLOWANCE

- Description: Modify the total reward distributed among the top 127 Super Representatives and Partners per round.
- Example: `[115_200_000_000] SUN` - which is 115,200 TRX.

- Range: `[0, 100_000_000_000_000_000] SUN`

7. CREATE_NEW_ACCOUNT_FEE_IN_SYSTEM_CONTRACT

- Description: Modify the fee for creating a new account via a system contract call.
- Example: `[0] SUN`.
- Range: `[0, 100_000_000_000_000_000] SUN`

8. CREATE_NEW_ACCOUNT_BANDWIDTH_RATE

- Description: Modify the Bandwidth consumption multiplier for creating a new account.
- Example: `[1]`.
- Range: `[0, 100_000_000_000_000_000]`

9. ALLOW_CREATION_OF_CONTRACTS

- Description: Enable or disable the creation of new smart contracts.
- Example: `True`
- Type: `True/False`

10. REMOVE_THE_POWER_OF_THE_GR

- Description: Remove the initial voting power held by the Genesis Representatives (GRs).
- Example: `True`
- Type: `True/False` - Notice: cannot be set back to `False` from `True`.

11. ENERGY_FEE

- Description: Modify the fee rate for consuming Energy when an account's Energy is insufficient.
- Example: `[10] SUN`.
- Range: `[0, 100_000_000_000_000_000] SUN`

12. EXCHANGE_CREATE_FEE

- Description: Modify the fee for creating a new TRC-10 trading pair in the decentralized exchange.
- Example: `[1_024_000_000] SUN` - which is 1024 TRX.
- Range: `[0, 100_000_000_000_000_000] SUN`

13. MAX_CPU_TIME_OF_ONE_TX

- Description: Modify the maximum CPU execution time allowed for a single transaction.
- Example: `[50] ms`.
- Range: `[0, 1000] ms`

14. ALLOW_UPDATE_ACCOUNT_NAME

- Description: Enable or disable the ability for accounts to update their name.
- Example: `False`
- Type: `True/False`

15. ALLOW_SAME_TOKEN_NAME

- Description: Allow different TRC-10 tokens to be created with the same name.
- Example: `True`
- Type: `True/False`

16. ALLOW_DELEGATE_RESOURCE

- Description: Enable or disable the resource delegation functionality (staking v1.0).
- Example: `True`
- Type: `True/False`

18. ALLOW_TVM_TRANSFER_TRC10

- Description: Allow smart contracts to natively transfer TRC-10 tokens using the `transferToken` call.
- Example: `True`
- Type: `True/False`

19. TOTAL_CURRENT_ENERGY_LIMIT

- Description: Modify the current total amount of Energy available on the network.
- Example: `[50_000_000_000]`.
- Range: `[0, 100_000_000_000_000_000]`

20. ALLOW_MULTI_SIGN

- Description: Enable or disable the multi-signature feature for accounts.
- Example: `True`
- Type: `True/False`

21. ALLOW_ADAPTIVE_ENERGY

- Description: Enable or disable the adaptive Energy model.
- Example: `True`
- Type: `True/False`

22. UPDATE_ACCOUNT_PERMISSION_FEE

- Description: Modify the fee for updating an account's permission structure.

- Example: `[100_000_000] SUN` - which is 100 TRX.
- Range: `[0, 100_000_000_000] SUN`

23. MULTI_SIGN_FEE

- Description: Modify the additional fee charged for processing a multi-signature transaction.
- Example: `[1_000_000] SUN` - which is 1 TRX.
- Range: `[0, 100_000_000_000] SUN`

24. ALLOW_PROTO_FILTER_NUM

- Description: Enable or disable protocol message filtering.
- Example: `False`
- Type: `True/False`

26. ALLOW_TVM_CONSTANTINOPLE

- Description: Enable the features of the Constantinople Ethereum upgrade in the TVM.
- Example: `True`
- Type: `True/False`

29. ADAPTIVE_RESOURCE_LIMIT_MULTIPLIER

- Description: Modify the multiplier used in the adaptive resource model.
- Example: `[1000]`.
- Range: `[1, 10000]`

30. ALLOW_CHANGE_DELEGATION

- Description: Allow users to change the recipient of their delegated resources without un-staking.
- Example: `True`
- Type: `True/False`

31. WITNESS_127_PAY_PER_BLOCK

- Description: Modify the block production reward for the top 28-127 ranked Partners.
- Example: `[160_000_000] SUN` - which is 160 TRX.
- Range: `[0, 100_000_000_000_000_000] SUN`

32. ALLOW_TVM_SOLIDITY_059

- Description: Enable TVM support for contracts compiled with Solidity v0.5.9 and above.
- Example: `True`
- Type: `True/False`

33. ADAPTIVE_RESOURCE_LIMIT_TARGET_RATIO

- Description: Modify the target ratio used in the adaptive resource model algorithm.
- Example: `[10]`.
- Range: `[1, 1000]`

35. FORBID_TRANSFER_TO_CONTRACT

- Description: Forbid direct TRX transfers to a contract without a payable fallback function.
- Example: `True`
- Type: `True/False`

39. ALLOW_SHIELDED_TRC20_TRANSACTION

- Description: Enable or disable privacy-preserving shielded transactions for TRC-20 tokens.
- Example: `True`
- Type: `True/False`

40. ALLOW_PBFT

- Description: Enable or disable the PBFT consensus mechanism for faster transaction finality.
- Example: `True`
- Type: `True/False`

41. ALLOW_TVM_ISTANBUL

- Description: Enable the features of the Istanbul Ethereum upgrade in the TVM.
- Example: `True`
- Type: `True/False`

44. ALLOW_MARKET_TRANSACTION

- Description: Enable or disable market order transactions on the decentralized exchange.
- Example: `True`
- Type: `True/False`

45. MARKET_SELL_FEE

- Description: Modify the fee charged for executing a sell order on the decentralized exchange.
- Example: `[0] SUN`.
- Range: `[0, 10_000_000_000] SUN`

46. MARKET_CANCEL_FEE

- Description: Modify the fee for cancelling an open order on the decentralized exchange.

- Example: `[0] SUN`.
- Range: `[0, 10_000_000_000] SUN`

47. MAX_FEE_LIMIT

- Description: Modify the maximum `fee_limit` a user can set for a single transaction.
- Example: `[15_000_000_000] SUN` - which is 15,000 TRX.
- Range: `[0, 100_000_000_000_000_000] SUN`

48. ALLOW_TRANSACTION_FEE_POOL

- Description: Enable or disable the transaction fee pool mechanism.
- Example: `False`
- Type: `True/False`

49. ALLOW_BLACKHOLE_OPTIMIZATION

- Description: Enable or disable an optimization where burning tokens consumes minimal Energy.
- Example: `True`
- Type: `True/False`

51. ALLOW_NEW_RESOURCE_MODEL

- Description: Enable or disable the new resource model (Staking 2.0).
- Example: `True`
- Type: `True/False`

52. ALLOW_TVM_FREEZE

- Description: Enable or disable TVM instructions related to the new staking/resource model (Staking 2.0).
- Example: `True`
- Type: `True/False`

53. ALLOW_ACCOUNT_ASSET_OPTIMIZATION

- Description: Enable an optimization for how account asset (TRC-10) balances are stored.
- Example: `True`
- Type: `True/False`

59. ALLOW_TVM_VOTE

- Description: Enable or disable the ability for smart contracts to execute voting operations.
- Example: `True`
- Type: `True/False`

60. ALLOW_TVM_COMPATIBLE_EVM

- Description: Enable or disable TVM compatibility features with the EVM.
- Example: `True`
- Type: `True/False`

61. FREE_NET_LIMIT

- Description: Modify the amount of free Bandwidth Points each account receives daily.
- Example: `[5000]`.
- Range: `[0, 100000]`

62. TOTAL_NET_LIMIT

- Description: Modify the total Bandwidth supplied by the network from staked TRX.
- Example: `[43_200_000_000]`.
- Range: `[0, 1_000_000_000_000]`

63. ALLOW_TVM_LONDON

- Description: Enable the features of the London Ethereum upgrade in the TVM.
- Example: `True`
- Type: `True/False`

65. ALLOW_HIGHER_LIMIT_FOR_MAX_CPU_TIME_OF_ONE_TX

- Description: Allow proposing a higher maximum value for the `MAX_CPU_TIME_OF_ONE_TX` parameter.
- Example: `True`
- Type: `True/False`

66. ALLOW_ASSET_OPTIMIZATION

- Description: Enable or disable account asset storage optimization.
- Example: `True`
- Type: `True/False`

67. ALLOW_NEW_REWARD

- Description: Enable or disable the new reward algorithm for SRs and voters.
- Example: `True`
- Type: `True/False`

68. MEMO_FEE

- Description: Modify the fee charged per byte for including a memo in a transaction.

- Example: `[1_000_000] SUN` - which is 1 TRX.
- Range: `[0, 1_000_000_000] SUN`

69. ALLOW_DELEGATE_OPTIMIZATION

- Description: Enable or disable optimizations for resource delegation storage.
- Example: `True`
- Type: `True/False`

70. UNFREEZE_DELAY_DAYS

- Description: Modify the number of days an asset is locked when using the new staking mechanism.
- Example: `[14] Days`.
- Range: `[1, 365] Days`

71. ALLOW_OPTIMIZED_RETURN_VALUE_OF_CHAIN_ID

- Description: Enable an optimization for the `CHAINID` opcode's return value.
- Example: `True`
- Type: `True/False`

72. ALLOW_DYNAMIC_ENERGY

- Description: Enable or disable the dynamic Energy model.
- Example: `True`
- Type: `True/False`

73. DYNAMIC_ENERGY_THRESHOLD

- Description: Modify the contract Energy consumption threshold before the dynamic penalty applies.
- Example: `[5_000_000_000]`.
- Range: `[0, 9223372036854775807]`

74. DYNAMIC_ENERGY_INCREASE_FACTOR

- Description: Modify the percentage factor by which Energy cost increases for a contract exceeding the threshold.
- Example: `[2000]`.
- Range: `[0, 10000]`

75. DYNAMIC_ENERGY_MAX_FACTOR

- Description: Modify the maximum multiplier for Energy cost under the dynamic Energy model.
- Example: `[34000]`.

- Range: `[0, 100000]`

76. ALLOW_TVM_SHANGHAI

- Description: Enable the features of the Shanghai Ethereum upgrade in the TVM.
- Example: `True`
- Type: `True/False`

77. ALLOW_CANCEL_ALL_UNFREEZE_V2

- Description: Enable the ability to cancel all pending unstaking requests under Staking 2.0.
- Example: `True`
- Type: `True/False`

78. MAX_DELEGATE_LOCK_PERIOD

- Description: Modify the maximum lock-up period for resource delegation.
- Example: `[864000]`.
- Range: `(86400, 10512000]`

79. ALLOW_OLD_REWARD_OPT

- Description: Enable an optimization for the reward withdrawal algorithm for the old reward system.
- Example: `True`
- Type: `True/False`

81. ALLOW_ENERGY_ADJUSTMENT

- Description: Enable or disable Energy adjustment functionalities.
- Example: `True`
- Type: `True/False`

82. MAX_CREATE_ACCOUNT_TX_SIZE

- Description: Modify the maximum transaction size for creating a new account.
- Example: `[1000]`.
- Range: `[500, 10000]`

83. ALLOW_TVM_CANCUN

- Description: Enable the features of the Cancun Ethereum upgrade in the TVM.
- Example: `True`
- Type: `True/False`

87. ALLOW_STRICT_MATH

- Description: Migrate TVM math operations to `java.lang.StrictMath` for cross-platform consistency.
- Example: `True`
- Type: `True/False`

88. CONSENSUS_LOGIC_OPTIMIZATION

- Description: Enable or disable optimizations in the consensus logic.
- Example: `True`
- Type: `True/False`

89. ALLOW_TVM_BLOB

- Description: Enable or disable TVM support for blob transactions (EIP-4844).
- Example: `True`
- Type: `True/False`