



TRON: Advanced Decentralized Blockchain Platform

Whitepaper Version: 2.2
TRON Protocol Version: 4.8.2.3

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	Resource Model	13
5.1	Resource Acquisition and Delegation	13
5.2	Bandwidth Model	13
5.3	Energy Model	14
6	Block	16
6.1	Block Header	16
6.1.1	Raw Data	16
6.1.2	Witness Signature	16
6.1.3	Block ID	16
6.2	Transaction	17
6.2.1	Signing	17
6.2.2	Fee	17
6.2.3	Transaction as Proof of Stake (TaPoS)	17
6.2.4	Transaction Confirmation	17
6.2.5	Structure	18
7	TRON Virtual Machine (TVM)	20
7.1	Introduction	20
7.2	Workflow	20
7.3	Performance	20
7.3.1	Lightweight Architecture	20
7.3.2	Robust	20
7.3.3	High Compatibility	20
7.3.4	Low Cost	20
8	Smart Contract	22
8.1	Introduction	22
8.2	Deployment	22
8.3	Trigger Function	22
8.4	TRON Solidity	22
9	Token	23
9.1	TRC-10 Token	23
9.2	TRC-20 Token	23
9.3	Beyond	23
10	Governance	24
10.1	Super Representative	24
10.1.1	General	24
10.1.2	Election	24
10.1.3	Reward	24
10.2	Committee	26
10.2.1	General	26
10.2.2	Dynamic Network Parameters	26
10.2.3	Create Proposal	26
10.2.4	Vote Proposal	26
10.2.5	Cancel Proposal	26
10.3	Structure	26
11	DApp Development	28
11.1	APIs	28
11.2	Networks	28

11.3 Resources	28
12 Conclusion	29
Appendix A: Terminology	30
Appendix B: Dynamic Network Parameters	33

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 inception¹.

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.

¹Previous whitepaper versions. V1.0: https://tron.network/static/doc/white_paper_v.1.0.pdf; V2.0: https://tron.network/static/doc/white_paper_v.2.0.pdf; V2.1: https://tron.network/static/doc/white_paper_v.2.1.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 saw continued growth in network accounts and stablecoin activity. TRON's total account count exceeded 100 million by June 2022 and 397 million by August 2026. Over the same period, the supply of TRC-20 USDT on TRON exceeded 91 billion. This growth was accompanied by continued improvements to platform stability, scalability, performance, and the network's economic model.

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.

Following these releases, the GreatVoyage-v4.8.1 series (Democritus and Hypatia) added governance-gated support for EIP-6780-aligned SELFDESTRUCT behavior, expanded ARM64 compatibility through JDK 17, and strengthened TVM edge-case handling. The GreatVoyage-v4.8.2 series (Pyrrho, Heraclitus and Parmenides) added governance-gated implementation support for selected Ethereum Pectra and Osaka features, strengthened resource accounting, refined contract-deployment validation and TVM edge-case handling, and improved networking and API performance. Overall, this period combined sustained growth in network accounts and stablecoin activity with continued advances in EVM compatibility, protocol robustness, resource management, and node operations.

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.

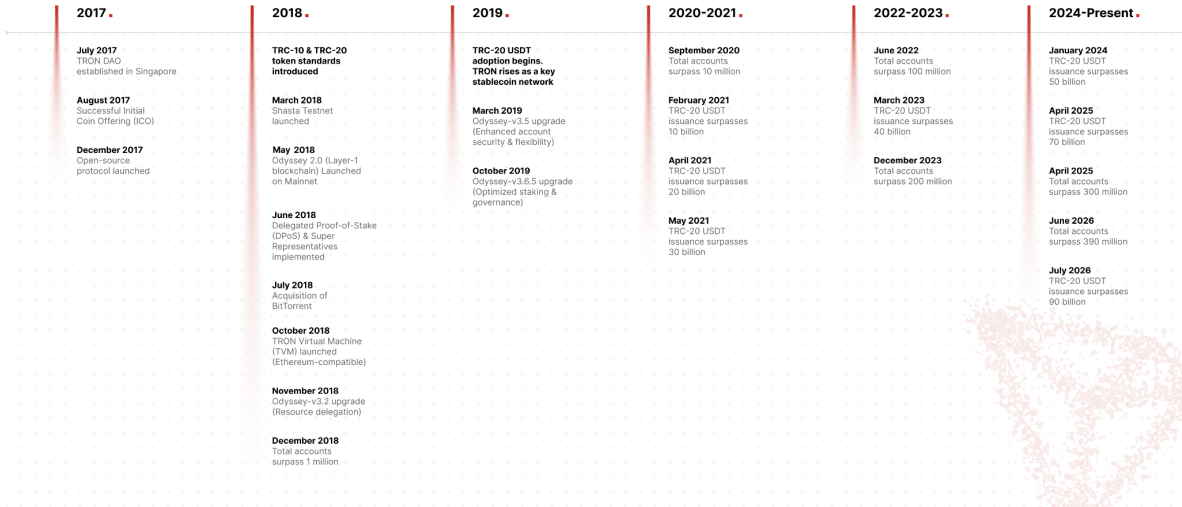


Figure 1: *TRON Development Timeline & Milestones (2017–Present)*

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, Solidity² 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. Blocks, accounts and contract state are persisted in a local embedded key-value store (see 2.2.1), and state updates are applied through a revocable snapshot layer so that transaction processing can be rolled back and chain switches handled safely (see 2.2.2).

2.2.1 Blockchain Storage

TRON blockchain storage supports two database engines, LevelDB and RocksDB. LevelDB, the default on x86-64 platforms, 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. On ARM64 platforms, only RocksDB is available.

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. It tracks how those blocks relate to one another so that a node can

²Solidity documentation: <https://docs.soliditylang.org/>.

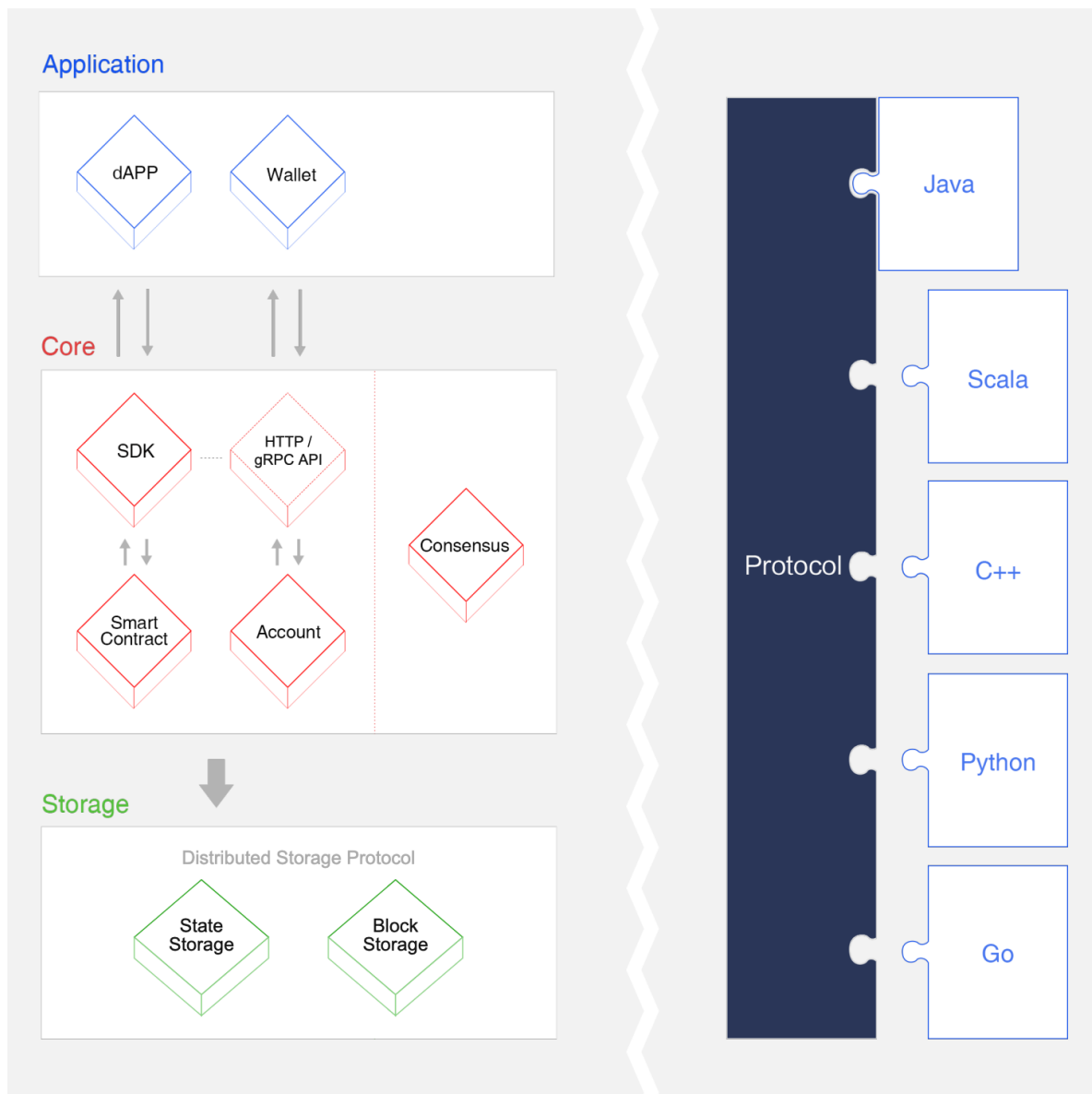


Figure 2: *TRON 3-layer Architecture*

find where two competing chains diverge. This enables any FullNode to switch swiftly from its own active chain into a new main chain.

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

The TRON protocol adheres to Google Protocol Buffers³, 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.

³Protocol Buffers documentation: <https://protobuf.dev/>.

3 Consensus

3.1 Delegated Proof of Stake (DPoS)

Proof of Work (PoW) was the earliest widely deployed blockchain consensus mechanism. It remains in use by Bitcoin⁴ and was used by Ethereum until Ethereum transitioned to Proof of Stake on September 15, 2022⁵. In PoW systems, miners assemble transactions into candidate blocks. The transactions are hashed pairwise into a Merkle tree whose root commits to the block's contents. Separately, miners vary the nonce and other header fields until the block-header hash satisfies the network's difficulty target. This proof-of-work search produces a valid candidate block; its confirmation depth then increases as additional valid blocks extend the chain. TRON uses DPoS rather than PoW, and its block header contains neither difficulty nor nonce fields:

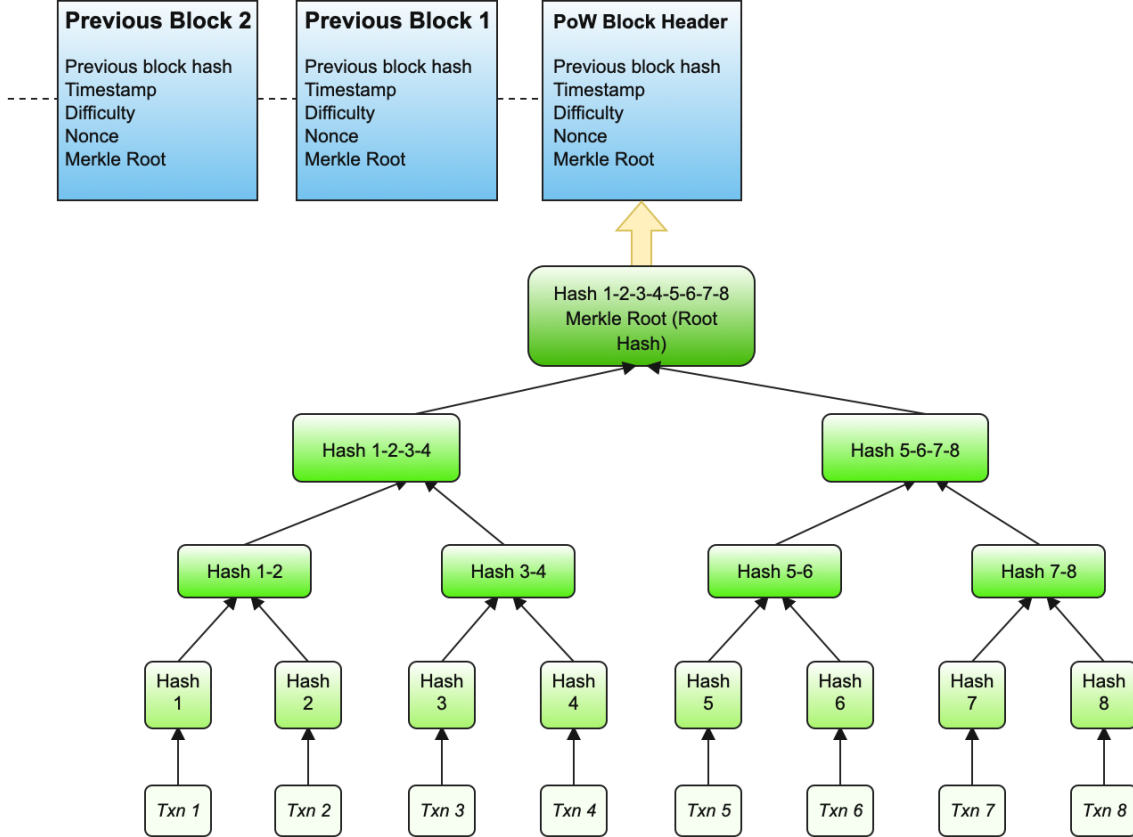


Figure 3: *PoW block header and transaction Merkle tree. The Merkle root commits to the transactions; the difficulty and nonce fields shown here are absent from TRON's DPoS block header.*

Cryptographic hashing algorithms are useful in network attack prevention because they possess several properties⁶:

- **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.

⁴Bitcoin whitepaper: <https://bitcoin.org/bitcoin.pdf>.

⁵Ethereum Mainnet Merge announcement: <https://blog.ethereum.org/2022/08/24/mainnet-merge-announcement>.

⁶PAAR, C., PELZL, J., Understanding Cryptography: A Textbook for Students and Practitioners, 2010 ed. Springer-Verlag Berlin Heidelberg, 2010.

- **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 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. TRX holders who stake TRX can vote for SR candidates. Votes are counted every six hours, and the top 27 candidates become the active SRs for the next cycle. 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. TRON's DPoS design combines a three-second block interval with a rotating set of 27 active block producers.

The TRON network targets a three-second block interval. Each successfully produced block currently generates an 8 TRX block-production reward for the producing SR and a 128 TRX voting-reward pool distributed among the top 127 candidates in proportion to the votes they receive. Rewards accrue automatically and can be withdrawn to each recipient's account balance once every 24 hours. The block-production reward and voting reward are governed by chain parameter #5 (`getWitnessPayPerBlock`) and chain parameter #31 (`getWitness127PayPerBlock`), respectively, and may be changed through governance. For detailed reward calculations, including maintenance-boundary skipped slots and distribution rules, see Section 10.1.3.

The current java-tron client supports two primary node configurations: `FullNode` and `Lite FullNode`. SRs run `FullNodes` in witness mode to produce blocks. `FullNodes` validate and synchronize full blockchain data and expose HTTP and gRPC APIs. `Lite FullNodes` start from reduced state snapshots and provide limited historical queries by default.

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.

A TRON Mainnet account address controlled by a `secp256k1` key pair is derived from its public key. Serialize the public key in uncompressed form and omit the leading `0x04` byte, obtaining the 64-byte `x || y` representation. Compute its Keccak-256 hash and take the rightmost 20 bytes. Prepend the single byte `0x41` to produce the 21-byte binary address. To produce the Base58Check representation, compute `SHA-256(SHA-256(address))`, append the first four bytes of that result as the checksum, and Base58-encode the resulting 25-byte sequence. The encoded Mainnet address begins with `T` and is 34 Base58 characters long. Smart contract addresses are not derived from a public key: a contract deployed by `CreateSmartContract` takes the last 20 bytes of `Keccak-256(transaction ID || creator address)`, and contracts created inside the TVM (`CREATE/CREATE2`) follow their own derivation rules; the same `0x41` prefix and Base58Check encoding then apply.

4.3 Structure

The three different account types are Normal, AssetIssue, and Contract. An Account contains more than forty fields; the seven below are a simplified subset that introduces the core concepts:

- **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, `TAcJvM...q4Xn5`;
- **balance:** The TRX balance of the account in sun – for example, `4213312` (4.213312 TRX);
- **votes:** Votes cast by the account for Super Representative candidates, each entry pairing a candidate address with the vote count – for example, `{("TBaPqR...x7Kc2",323), ("TCmNsV...p9Yw4",88), ..., ("TDbHdQ...r6Zk3",10001)}`;

- **asset**: Other assets, besides TRX, held in the account – for example, {<"WishToken", 66666>, <"Dogie", 233>}. This is the legacy map, keyed by token name; the current **assetV2** field holds the same balances keyed by token ID.
- **latest_opration_time**: The timestamp of the account's most recent operation. The field name is misspelled in the protocol definition and is reproduced here as it appears in `core/Tron.proto`.

Protobuf data structure:

```

1 message Account { // simplified subset of fields, see core/Tron.proto
2     message Vote {
3         bytes vote_address = 1;
4         int64 vote_count = 2;
5     }
6
7     bytes account_name = 1;
8
9     AccountType type = 2;
10    bytes address = 3;
11    int64 balance = 4;
12    repeated Vote votes = 5;
13    map<string, int64> asset = 6;
14
15    int64 latest_opration_time = 10; // sic: misspelled in core/Tron.proto
16 }
17 enum AccountType {
18     Normal = 0;
19     AssetIssue = 1;
20     Contract = 2;
21 }

```

5 Resource Model

TRON uses Bandwidth and Energy for transaction processing and smart contract execution. Transactions consume Bandwidth in proportion to their size in bytes, while smart contract execution consumes Energy according to the TVM operations performed. Accounts can obtain Bandwidth or Energy by staking TRX or by receiving delegated resources from another account. Each activated account also receives a daily free Bandwidth allowance. When available resources are insufficient, TRX is consumed according to governance-controlled fee parameters. TRON Power, which is used for voting, is described in Chapter 10.

5.1 Resource Acquisition and Delegation

Accounts obtain Bandwidth and Energy by staking TRX under Stake 2.0 or by receiving delegated resources from another account. Resource delegation allows another account to use the corresponding Bandwidth or Energy without transferring ownership of the underlying staked TRX.

A delegation may be subject to a lock period, and the delegated balance must be undelegated before the corresponding staked TRX can be unstaked. The relevant contract types and field-level rules, including partial undelegation and the treatment of unrecovered resource usage, are described in the notes that follow the contract-type list in Section 6.2.5.

5.2 Bandwidth Model

Ordinary transactions consume only Bandwidth, while smart contract transactions consume both Energy and Bandwidth. Users can obtain Bandwidth or Energy by staking TRX or by receiving delegated resources from another account. A daily free Bandwidth allowance is also available.⁷

When a TRX transaction is broadcast, it is transmitted and stored in the form of a byte array over the network. Bandwidth consumption is based on the transaction's serialized size: conceptually, a transaction of 200 bytes consumes about 200 Bandwidth. In the current java-tron implementation, when TVM support is enabled, the charged size also includes a protocol-defined result allowance of 64 bytes per contract, so the exact amount can be slightly higher than the raw byte length. However, if a TRX or token transfer results in the target account being created, then only the Bandwidth consumed to create the account will be deducted, and additional Bandwidth will not be deducted. A TRX or TRC-10 transfer to an inactive address currently incurs a governance-controlled account-creation fee of 1 TRX. If the sender also lacks sufficient staked Bandwidth, an additional 0.1 TRX Bandwidth fee is charged. Account-creation transactions are additionally subject to a separate size-validation rule (currently 1000 bytes, chain parameter #82, `getMaxCreateAccountTxSize`), which is independent of the Bandwidth charge itself. In an account creation scenario, the network will first consume the Bandwidth that the transaction initiator obtained by staking 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 gained by the transaction initiator by staking TRX. If that is insufficient, it then consumes from the daily free Bandwidth. 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 byte⁸. Thus, for most TRX holders who may not necessarily stake their TRX to participate in SR voting, the first step is automatically skipped (since staked TRX balance = 0) and the daily free Bandwidth powers the transaction.

For a TRC-10 token transfer to an existing account, the network first attempts to use Bandwidth sponsored by the token issuer, subject to the issuer's available Bandwidth and the per-account and total free-Bandwidth limits configured for that token. Otherwise, it consumes the sender's Bandwidth obtained by staking TRX or received through delegation, then the sender's daily free Bandwidth,

⁷The specific amount of this free allowance is defined by chain parameter #61 (`getFreeNetLimit`). TRON Chain Parameters API: <https://developers.tron.network/reference/wallet-getchainparameters>.

⁸This cost is defined by chain parameter #3 (`getTransactionFee`). TRON Chain Parameters API: <https://developers.tron.network/reference/wallet-getchainparameters>.

and finally burns the sender's TRX for the transaction's full Bandwidth charge. Each source must be able to cover the whole transaction on its own; partial deductions from several sources are not combined. Transfers that create a new account follow the separate account-creation charging path described above.

5.3 Energy Model

Smart-contract execution consumes Energy. Accounts obtain Energy by staking TRX or receiving delegated Energy. If the available Energy is insufficient, TRX may be burned to cover the remainder, subject to the account balance and the transaction's `fee_limit`.

The Energy allocated per staked TRX is calculated dynamically:

$$\text{Energy per staked TRX} = \frac{\text{TotalEnergyCurrentLimit}}{\text{TotalEnergyWeight}}$$

`TotalEnergyCurrentLimit` is governed by chain parameter #19 (`getTotalEnergyCurrentLimit`), while `TotalEnergyWeight` represents the total TRX staked for Energy across the network. Consumed staked or delegated Energy is restored progressively over a rolling 24-hour window.

For a smart-contract deployment or trigger transaction, let:

- E be the caller's remaining staked or delegated Energy;
- B be the caller's available balance in sun after deducting any `call_value`;
- L be the transaction's `fee_limit` in sun;
- P be the Energy price in sun per Energy, defined by chain parameter #11 (`getEnergyFee`).

The caller's maximum usable Energy is:

$$\text{CallerEnergyLimit} = \min \left(E + \frac{B}{P}, \frac{L}{P} \right)$$

For contract deployment, `CallerEnergyLimit` is the maximum Energy budget available for execution.

When an existing contract is triggered, the contract developer may also contribute Energy, subject to `consume_user_resource_percent`, `origin_energy_limit`, and the developer's remaining Energy:

$$\text{TransactionEnergyLimit} = \text{CallerEnergyLimit} + \text{DeveloperEnergyLimit}$$

`TransactionEnergyLimit` is the maximum Energy budget available for that contract execution. The transaction is charged only for the Energy actually consumed.

The Dynamic Energy Model, specified in TIP-491 and activated on Mainnet on February 5, 2023 following governance proposal No. 83, adjusts a contract's Energy consumption according to its usage during previous maintenance cycles:

$$\text{Adjusted Energy usage} = \text{Basic Energy usage} \times (1 + \text{energy_factor})$$

Contracts with sustained high usage receive a higher `energy_factor`, up to a governance-defined maximum; the factor decreases as usage falls. This mechanism changes how quickly the available transaction Energy budget is consumed and helps prevent a small number of popular contracts from consuming a disproportionate share of network resources. In this formula, `energy_factor` denotes the normalized additional factor. Raw chain parameters #74 (`getDynamicEnergyIncreaseFactor`) and #75 (`getDynamicEnergyMaxFactor`) use a precision factor of 10,000, so the current values 2,000 and 34,000 represent a 20% increase rate per cycle and a maximum additional factor of 3.4,

respectively. The resulting maximum total multiplier is 4.4 for execution costs to which the factor applies. The factor is applied per contract, not to every fee in a multi-contract transaction.

6 Block

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

```
1 message Block {
2
3     repeated Transaction transactions = 1;
4     BlockHeader block_header = 2;
5
6 }
```

6.1 Block Header

A block header contains two fields: **raw_data** and **witness_signature**. Protobuf data structure:

```
1 message BlockHeader {
2     message raw {
3         int64 timestamp = 1;
4         bytes txTrieRoot = 2;
5         bytes parentHash = 3;
6         int64 number = 7;
7         int64 witness_id = 8;
8         bytes witness_address = 9;
9         int32 version = 10;
10        bytes accountStateRoot = 11;
11    }
12    raw raw_data = 1;
13    bytes witness_signature = 2;
14 }
```

6.1.1 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, 7dac5a...3ed.
3. **parentHash**: the Block ID of the parent block (the composite identifier defined in Section 6.1.3) – for example, 000000000046c7f3...3ed.
4. **number**: the block height – for example, 4638708.
5. **version**: the block protocol version written by the producing SR; the network uses it to track SR upgrades and decide when a hard fork is activated – for example, 37.
6. **witness_address**: the address of the SR that produced this block – for example, 41928c...4d21.
7. **witness_id**: a legacy int64 field that is not populated by current java-tron (always 0); the producing SR is identified by witness address – for example, 0.
8. **accountStateRoot**: the root hash of the account-state Merkle (trie) tree after this block has been applied — for example, 0x7dac...3ed.

6.1.2 Witness Signature

The Protobuf field **witness_signature** contains the producing SR’s signature for the block header.

6.1.3 Block ID

A block ID uniquely identifies a block. It is the SHA-256 hash of the block header’s raw data, with the first 8 bytes replaced by the block height encoded as a big-endian 64-bit integer. The height can therefore be read directly from this composite identifier.

ChainInventory.BlockId is a separate structure used in network sync messages. It contains 2 parameters:

1. **hash**: the block ID.
2. **number**: the block height.

6.2 Transaction

6.2.1 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 resulting hash is signed by the private key or keys authorized under the initiating account's selected permission: the owner permission by default, or an active permission selected by `PermissionId`. The signatures must satisfy that permission's weight threshold and operation restrictions. The signature result is then added to the transaction.

6.2.2 Fee

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 chain parameter #4 (`getAssetIssueFee`)
2. Apply to be an SR candidate: 9,999 TRX — see chain parameter #1 (`getAccountUpgradeCost`).
3. Create a Bancor transaction: 1,024 TRX — see chain parameter #12 (`getExchangeCreateFee`).
4. Update the account permission: 100 TRX — see chain parameter #22 (`getUpdateAccountPermissionFee`).
5. Activate the account: 1 TRX — see chain parameter #7 (`getCreateNewAccountFeeInSystemContract`); insufficient Bandwidth obtained through staking in the creator's account incurs an additional 0.1 TRX fee to pay for the Bandwidth — see chain parameter #2 (`getCreateAccountFee`).
6. Multisig transaction: 1 TRX — see chain parameter #23 (`getMultiSignFee`).
7. Transaction note: 1 TRX — see chain parameter #68 (`getMemoFee`).

6.2.3 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.

6.2.4 Transaction Confirmation

A transaction is included in a block after being broadcast to the network. The transaction is considered confirmed (solidified) once at least 19 of the 27 active Super Representatives, counting the SR that produced the block itself, have each produced a block at or above its height. In practice this usually takes about 19 subsequent blocks.

TRON schedules one block-production slot approximately every 3 seconds. If the scheduled SR misses its slot, the observed interval between consecutive blocks can be longer. In general, a transaction is considered fully confirmed after approximately 1 minute.

6.2.5 Structure

The Transaction.Contract.ContractType enumeration includes the following contract types (excerpt):

```

1 message Transaction {
2   message Contract {
3
4     enum ContractType { // subset; see core/Tron.proto for the complete list
5       AccountCreateContract = 0; // Create account/wallet
6       TransferContract = 1; // Transfer TRX
7       TransferAssetContract = 2; // Transfer TRC-10 token
8       VoteWitnessContract = 4; // Vote for SR
9       WitnessCreateContract = 5; // Create a new SR account
10      AssetIssueContract = 6; // Create a new TRC-10 token
11      WitnessUpdateContract = 8; // Update SR information
12      ParticipateAssetIssueContract = 9; // Purchase TRC-10 token
13      AccountUpdateContract = 10; // Update account/wallet information
14
15      FreezeBalanceContract = 11; // Stake 1.0: Stake TRX for Bandwidth or Energy (
legacy; new requests disabled)
16      UnfreezeBalanceContract = 12; // Stake 1.0: Unstake TRX from a legacy staking
position
17
18      WithdrawBalanceContract = 13; // Withdraw SR rewards, once per day
19      UnfreezeAssetContract = 14; // Unfreeze TRC-10 token
20      UpdateAssetContract = 15; // Update a TRC-10 token's information
21      ProposalCreateContract = 16; // Create a new network proposal by any SR
22      ProposalApproveContract = 17; // SR votes yes for a network proposal
23      ProposalDeleteContract = 18; // Delete a network proposal by owner
24      CreateSmartContract = 30; // Deploy a new smart contract
25      TriggerSmartContract = 31; // Call a function on a smart contract
26      GetContract = 32; // Get an existing smart contract
27      UpdateSettingContract = 33; // Update a smart contract's parameters
28      ExchangeCreateContract = 41; // Create a token trading pair on DEX
29      ExchangeInjectContract = 42; // Inject funding into a trading pair
30      ExchangeWithdrawContract = 43; // Withdraw funding from a trading pair
31      ExchangeTransactionContract = 44; // Perform token trading
32      UpdateEnergyLimitContract = 45; // Update origin_energy_limit on a smart
contract
33      AccountPermissionUpdateContract = 46; // Update account permissions
34      ClearABIContract = 48; // Clear a smart contract's ABI
35      UpdateBrokerageContract = 49; // Update SR brokerage commission rate
36      MarketSellAssetContract = 52; // Place a market sell order
37      MarketCancelOrderContract = 53; // Cancel an existing market order
38
39      FreezeBalanceV2Contract = 54; // Stake 2.0: Stake TRX for Bandwidth or Energy
40      UnfreezeBalanceV2Contract = 55; // Stake 2.0: Initiate unstaking and enter the
waiting period
41      WithdrawExpireUnfreezeContract = 56; // Stake 2.0: Withdraw TRX after the
unstaking waiting period
42      DelegateResourceContract = 57; // Stake 2.0: Delegate Bandwidth or Energy to
another account
43      UnDelegateResourceContract = 58; // Stake 2.0: Reclaim delegated Bandwidth or
Energy
44      CancelAllUnfreezeV2Contract = 59; // Stake 2.0: Cancel all pending unstaking
requests
45
46    }
47    ContractType type = 1;
48    google.protobuf.Any parameter = 2;
49    bytes provider = 3;
50    bytes ContractName = 4;
51    int32 Permission_id = 5;
52  }
53 }
```

For Stake 2.0 resource delegation, `DelegateResourceContract` delegates Bandwidth or Energy to another account. Its `balance` field is denominated in sun and specifies the amount of staked TRX whose corresponding resource quota is delegated; it does not specify a number of Bandwidth units or Energy units. The `resource` field selects Bandwidth or Energy. The `lock` field determines whether the delegation is locked. When `lock` is `true`, `lock_period` specifies the lock duration in blocks. The receiving account must already exist, must not be the owner account, and must not be a contract account.

`UnDelegateResourceContract` supports partial reclamation by allowing the owner to specify a balance smaller than the total delegated balance. If the receiving account has consumed part of the delegated resource, the corresponding unrecovered usage is transferred back to the owner and continues through the normal recovery process. Consequently, the reclaimed resource quota may not be immediately available in full for further delegation. A staked balance associated with an active resource delegation cannot be unstaked until the corresponding resource has first been undelegated.

7 TRON Virtual Machine (TVM)

7.1 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 was initially derived from the EVM⁹ and remains compatible with the Solidity smart contract ecosystem. It operates within TRON's DPoS-based protocol architecture.

TVM execution consumes Energy. Energy allocation, recovery, and charging rules are described in Section 5.3.

7.2 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.

7.3 Performance

7.3.1 Lightweight Architecture

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

7.3.2 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 staking 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 while the TVM's defined base Energy cost schedule provides a predictable baseline for resource consumption.

7.3.3 High Compatibility

The TVM is largely compatible with the EVM, allowing most smart contracts to be ported directly without source-code modifications. However, developers should still review the documented TVM–EVM differences when migrating.

7.3.4 Low Cost

When sufficient staked or delegated Energy is available, smart-contract execution does not require the direct burning of TRX. The TVM also offers all-in-one interfaces for contract deployment, triggering, and viewing to offer the convenience for developers.

⁹Ethereum Virtual Machine (EVM): <https://ethereum.org/developers/docs/evm/>.

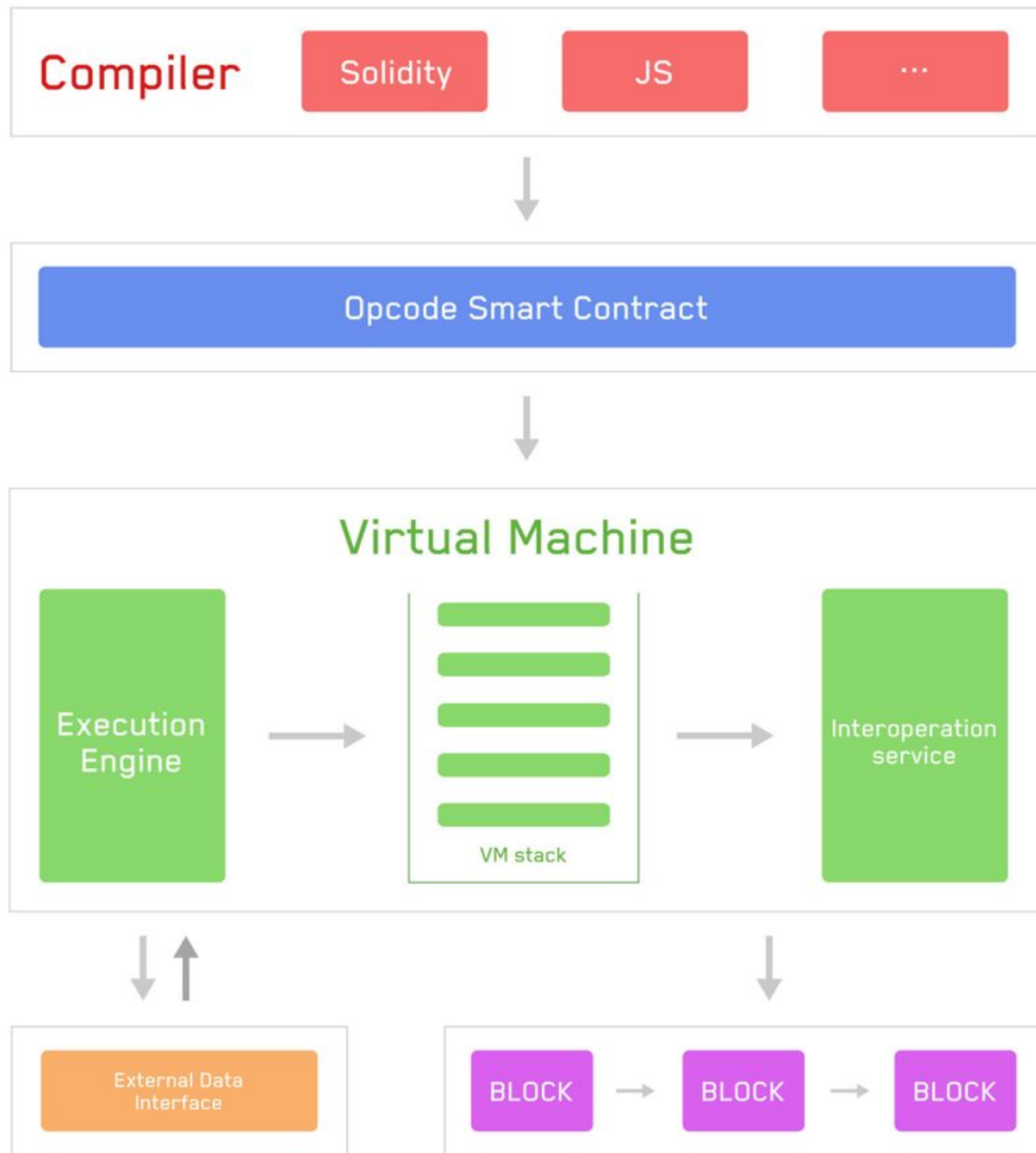


Figure 4: *TVM Workflow*

8 Smart Contract

8.1 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.

Deploying or triggering a state-changing smart contract consumes Energy for TVM execution and Bandwidth for transaction broadcast, as described in Sections 5.2 and 5.3.

8.2 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.

8.3 Trigger Function

Once the TRON smart contracts are deployed, their functions can be triggered individually either via TronIDE or through API calls. Function calls included in an on-chain transaction consume Energy, including calls to `view` or `pure` functions. Constant (off-chain) calls are executed locally by the node: Energy usage is still measured and execution limits apply, but the caller is not charged on-chain.

8.4 TRON Solidity

TRON Solidity is a fork from Ethereum's Solidity language, introducing broad support for the TRON ecosystem. This native integration encompasses TRX and sun units, TRC-10 tokens, the TRON resource model, and various other custom network extensions. TRON Solidity continuously aligns with upstream version releases. As of August 2026, the latest supported version is 0.8.31; for current and detailed version history, visit <https://github.com/tronprotocol/solidity/releases>.

9 Token

9.1 TRC-10 Token

In the TRON network, an account can issue one TRC-10 token, subject to the protocol's validation rules and the Asset Issue Fee (currently 1024 TRX, chain parameter #4). An account that has already issued a token cannot issue a second TRC-10 token. To issue a token, 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, the entire network's maximum daily token transfer Bandwidth, total token supply, locking duration in days, and the total amount of tokens locked.

9.2 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:

```
1 interface TRC20Interface {
2     function totalSupply() external view returns (uint);
3     function balanceOf(address tokenOwner) external view returns (uint balance);
4     function allowance(address tokenOwner, address spender) external view returns (uint
    remaining);
5     function transfer(address to, uint tokens) external returns (bool success);
6     function approve(address spender, uint tokens) external returns (bool success);
7     function transferFrom(address from, address to, uint tokens) external returns (bool
    success);
8     event Transfer(address indexed from, address indexed to, uint tokens);
9     event Approval(address indexed tokenOwner, address indexed spender, uint tokens);
10 }
```

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.

9.3 Beyond

Because the TVM is largely EVM-compatible and supports Solidity, many Ethereum token standards can be ported to TRON after compatibility checks.

10 Governance

10.1 Super Representative

10.1.1 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, the account can join the SR election.

10.1.2 Election

TRON Power (denoted as TP) is required for voting. Under the current Mainnet resource model, each TRX staked for Bandwidth or Energy contributes one TP.

TP is calculated in the following way:

$$1TP = 1TRX \text{ staked for Bandwidth or Energy}$$

Delegating Bandwidth or Energy does not transfer the underlying staked TRX or TRON Power; voting rights remain with the delegator.

If chain parameter #51 (`getAllowNewResourceModel`) is enabled, TP is separated from Bandwidth and Energy, and TRX may be staked specifically to obtain TP. In that case, the formula above no longer fully describes how TP is obtained.

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

Initiating unstaking reduces the account's TP, and the unstaked TRX becomes withdrawable after the 14-day waiting period. When unstaking lowers an account's TP, existing votes remain unchanged as long as the remaining TP is sufficient to cover them. The protocol's reclamation mechanism first deducts from the account's idle TP balance. In the event of insufficient idle TP, the system proceeds to reclaim used TP. For accounts maintaining active votes across multiple SRs, the required TP is revoked proportionally from each supported SR.

Note that a new `VoteWitnessContract` replaces the account's entire current vote allocation rather than adding to it. Earlier vote transactions remain on chain, and rewards already earned by previous votes are settled before the new allocation takes effect. The active SR set and the reward snapshot are updated at the next maintenance period.

10.1.3 Reward

a. Vote Reward

A voting reward pool of 128 TRX is generated for each successfully produced block and shared by the 27 SRs and 100 SR Partners in proportion to the votes each candidate receives. Because two block slots are skipped at each six-hour maintenance boundary, the nominal voting-reward pool is 921,344 TRX per maintenance period and 1,345,162,240 TRX per 365-day year, assuming no additional missed slots. The actual total depends on the number of blocks successfully produced and may change if the governing parameter is adjusted.

Total vote reward per round

Why 921,344 TRX every round?

$$921,344 \text{ TRX} = \text{total vote reward per round (VR/round)}$$

$$\text{VR/round} = 128 \text{ TRX/block} \times (20 \text{ scheduled block slots/min} \times 60 \text{ mins/hr} \times 6 \text{ hrs/round} - 2 \text{ skipped slots/round})$$

Note: the voting-reward rate is governed by chain parameter #31 (getWitness127PayPerBlock), currently set to 128 TRX per successfully produced block. The actual reward for a round is determined by the number of blocks successfully produced.

Total vote reward per year

Why 1,345,162,240 TRX every year?

$$1,345,162,240 \text{ TRX} = \text{total vote reward per year (VR/year)}$$

$$\text{VR/year} = 921,344 \text{ TRX/round} \times 4 \text{ rounds/day} \times 365 \text{ days/year}$$

b. Block Reward

Also known as Super Representative Reward, the block-production reward is paid to the SR that successfully produces each block. The current reward is 8 TRX per successfully produced block. Because two block slots are skipped at each six-hour maintenance boundary, the nominal network-wide block-production reward is 57,584 TRX per round and 84,072,640 TRX per 365-day year, assuming no additional missed slots.

Total block reward per round

Why 57,584 TRX every round?

$$57,584 \text{ TRX} = \text{total block reward per round (BR/round)}$$

$$\text{BR/round} = 8 \text{ TRX/block} \times (20 \text{ scheduled block slots/min} \times 60 \text{ mins/hr} \times 6 \text{ hrs/round} - 2 \text{ skipped slots/round})$$

Note: the block-production reward is governed by chain parameter #5 (getWitnessPayPerBlock), currently set to 8 TRX per successfully produced block. The actual reward for a round is determined by the number of blocks successfully produced.

Total block reward per year

Why 84,072,640 TRX every year?

$$84,072,640 \text{ TRX} = \text{total block reward per year (BR/year)}$$

$$\text{BR/year} = 57,584 \text{ TRX/round} \times 4 \text{ rounds/day} \times 365 \text{ days/year}$$

c. Reward Calculation

SR reward calculation

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

$$\text{VR} = \text{total VR} \times (\text{votes received by the SR} / \text{total votes received by the top 127 candidates})$$

$$\text{BR} = \text{number of blocks successfully produced by the SR} \times 8 \text{ TRX/block}$$

Equivalently, when the SR's scheduled and missed slots are known:

$$\text{BR} = (\text{scheduled slots assigned to the SR} - \text{slots missed by the SR}) \times 8 \text{ TRX/block}$$

Block-production rewards are credited for blocks actually produced; they are not calculated by first dividing the network-wide reward equally among 27 SRs. In addition, 7,198 nominally produced blocks per round cannot be divided evenly among 27 SRs.

Rank 28–127 SR Partner reward calculation

Gross reward allocated to an SR Partner before brokerage:

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

$$\text{VR} = \text{total VR} \times (\text{votes received by the SR Partner} / \text{total votes received by the top 127 candidates})$$

Note: The formulas above calculate the gross reward allocated to each SR or SR Partner for a round. Both block-production and voting rewards are subject to the recipient's brokerage ratio: the recipient retains the brokerage share, and the remainder is distributed among its voters in proportion to their votes.

10.2 Committee

10.2.1 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. At expiry, a proposal is approved if at least 18 active SRs have approvals on record, and the approved parameter changes take effect at the same maintenance-period boundary.

10.2.2 Dynamic Network Parameters

Please see Appendix B for details.

10.2.3 Create Proposal

Each SR, SR partner and SR candidate is entitled to initiate proposals. SRs also have the rights to vote for the proposals.

10.2.4 Vote Proposal

SRs, SR Partners, and SR candidates may submit or withdraw an approval before a proposal expires, but only approvals from active SRs at expiration are counted toward the outcome, as the protocol records approvals only. The base voting window is controlled by chain parameter #92 (`getProposalExpireTime`), currently set to three days, and expiration is set to the first maintenance-period boundary after that window. At expiration, the proposal passes if it has approvals from at least 18 active SRs; otherwise, it fails.

10.2.5 Cancel Proposal

The proposer can cancel a pending proposal before it expires.

10.3 Structure

In the protocol data model, each SR is represented by a `Witness` object containing 9 fields:

1. **address**: the address of this SR – for example, `TBaPqR...x7Kc2`.
2. **voteCount**: the number of votes received by this SR – for example, `234234`.
3. **pubKey**: a reserved public-key field. It is not populated when a `Witness` is created and is not used for block signature verification, which recovers the signer address from the block signature and compares it with the witness address (or the account's witness permission when multi-signature is enabled) – for example, empty.
4. **url**: the URL of this SR – for example, `https://www.noonetrust.com/`.

5. **totalProduced**: the number of blocks produced by this SR – for example, 2434.
6. **totalMissed**: the number of blocks missed by this SR – for example, 7.
7. **latestBlockNum**: the latest block height recorded for this SR – for example, 4522.
8. **latestSlotNum**: the latest block-production slot for this SR – for example, 4523.
9. **isJobs**: a boolean flag.

Protobuf data structure:

```
1 message Witness{
2     bytes address = 1;
3     int64 voteCount = 2;
4     bytes pubKey = 3;
5     string url = 4;
6     int64 totalProduced = 5;
7     int64 totalMissed = 6;
8     int64 latestBlockNum = 7;
9     int64 latestSlotNum = 8;
10    bool isJobs = 9;
11 }
```

11 DApp Development

11.1 APIs

The TRON network provides HTTP and gRPC API endpoints for interacting with the network via FullNodes; see the TRON Developer Hub API reference for the full list. 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 endpoints can be directed towards a local private network, the Shasta testnet, the Nile testnet, or the TRON Mainnet.

11.2 Networks

TRON has a Shasta testnet, a Nile testnet, as well as a Mainnet. Developers may connect through FullNode APIs offered by services such as TronGrid or, where the network permits, by operating their own nodes. Shasta currently does not accept third-party node operators; Nile and Mainnet follow their current deployment guidance. 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.

11.3 Resources

The TRON Developer Hub¹⁰ is a comprehensive API documentation site designed for 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.

¹⁰TRON Developer Hub: <https://developers.tron.network/>.

12 Conclusion

TRON combines DPoS consensus, a dual-resource model, and a largely EVM-compatible virtual machine to support high-throughput decentralized applications and on-chain asset transfers. Its protocol continues to evolve through on-chain governance and versioned client upgrades.

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 java-tron's protocol definitions (for example `AssetIssueContract`, `TransferAssetContract` and the Account `asset/assetV2` fields), "asset" refers specifically to native TRC-10 tokens. The broader term "token" also covers contract-based tokens such as TRC-20; see Chapter 9.

Bandwidth

To keep the network operating smoothly, TRON network transactions use Bandwidth as fuel. Each account gets a daily free amount of Bandwidth, defined by chain parameter #61 (`getFreeNetLimit`) in the TRON Chain Parameters API, and more can be obtained by staking TRX for Bandwidth. Both TRX and TRC-10 token transfers are normal transactions costing Bandwidth. Smart contract deployment and execution transactions consume both Bandwidth and Energy.

Block

Blocks contain the digital records of transactions. In the TRON protocol a Block message consists of two fields: `block_header` and the list of transactions (see Section 6).

Block Reward

Block production and voting rewards are tracked by the protocol's reward accounting: an SR's share (after brokerage) accrues to the allowance field of its own account, and voter rewards are computed per maintenance cycle (see Section 10.1.3). Withdrawing adds the claimable amount to the account's existing balance. 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, the block height, version, the producing SR's address, and the account state root.

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)

gRPC¹¹ 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 a 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 allows any FullNode to switch swiftly from its own active chain to 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. It remains the default storage engine, with RocksDB supported as an alternative. 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)

Public test networks that run separately from TRON Mainnet. Developers can use public API endpoints and faucets without using assets that have real-world value. Node-operation policies differ by network: Shasta currently does not accept third-party node operators. Testnet tokens have no value and anyone can request more from the public faucet.

remote procedure call (RPC)

In distributed computing, an RPC¹² 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.

¹¹gRPC documentation: <https://grpc.io/docs/what-is-grpc/>.

¹²Remote procedure call: https://en.wikipedia.org/wiki/Remote_procedure_call.

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.

sun

sun replaced drop as the smallest unit of TRX. $1 \text{ TRX} = 1,000,000 \text{ sun}$.

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 in the block header as a Unix epoch timestamp in milliseconds (elapsed since 00:00:00 01 Jan 1970 UTC). Inside the TVM, block.timestamp (the TIMESTAMP opcode) reports the same time in seconds, obtained by dividing the block header value by 1000.

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

The values below are illustrative examples and may differ from current Mainnet settings. For current values, query `/wallet/getchainparameters` or TRONSCAN. For activation-only Boolean parameters, governance proposals may set the value from `False` to `True`, but cannot subsequently set it back to `False`.

ID / Parameter / API Name	Description	Example / Range
#0 — MAINTENANCE.TIME.INTERVAL <code>getMaintenanceTimeInterval</code>	Modify the maintenance interval time, also the interval for SRs to calculate and distribute voting rewards.	Example: <code>[6 * 3600 * 1000]</code> ms - which is 6 hours. Range: <code>[3 * 27 * 1000, 24 * 3600 * 1000]</code> ms
#1 — ACCOUNT.UPGRADE.COST <code>getAccountUpgradeCost</code>	Modify the cost for an account to apply to become a Super Representative candidate.	Example: <code>[9.999.000.000]</code> sun - which is 9999 TRX. Range: <code>[0, 100.000.000.000.000.000]</code> sun
#2 — CREATE.ACCOUNT.FEE <code>getCreateAccountFee</code>	Modify the fee for creating a new account by sending TRX to a new address.	Example: <code>[100.000]</code> sun - which is 0.1 TRX. Range: <code>[0, 100.000.000.000.000.000]</code> sun
#3 — TRANSACTION.FEE <code>getTransactionFee</code>	Modify the fee rate for consuming Bandwidth when an account's free/staked Bandwidth is insufficient.	Example: <code>[1.000]</code> sun/byte. Range: <code>[0, 100.000.000.000.000.000]</code> sun/byte
#4 — ASSET.ISSUE.FEE <code>getAssetIssueFee</code>	Modify the one-time fee for issuing a new TRC-10 token.	Example: <code>[1.024.000.000]</code> sun - which is 1024 TRX. Range: <code>[0, 100.000.000.000.000.000]</code> sun
#5 — WITNESS.PAY.PER.BLOCK <code>getWitnessPayPerBlock</code>	Modify the reward paid to a Super Representative for successfully producing a block.	Example: <code>[8.000.000]</code> sun - which is 8 TRX. Range: <code>[0, 100.000.000.000.000.000]</code> sun
#6 — WITNESS.STANDBY.ALLOWANCE <code>getWitnessStandbyAllowance</code>	Modify the total reward distributed in proportion to votes among the 27 SRs and 100 SR Partners per maintenance period under the legacy reward mechanism. When chain parameter #30 (<code>getChangeDelegation</code>) is enabled, this legacy parameter is no longer used; the voting reward generated for each successfully produced block is instead governed by chain parameter #31 (<code>getWitness127PayPerBlock</code>).	Example: <code>[115.200.000.000]</code> sun - which is 115,200 TRX. Range: <code>[0, 100.000.000.000.000.000]</code> sun
#7 — CREATE.NEW.ACCOUNT.FEE.IN.SYSTEM.CONTRACT <code>getCreateNewAccountFeeInSystemContract</code>	Modify the fee for creating a new account via a system contract call.	Example: <code>[1.000.000]</code> sun - which is 1 TRX. Range: <code>[0, 100.000.000.000.000.000]</code> sun
#8 — CREATE.NEW.ACCOUNT.BANDWIDTH.RATE <code>getCreateNewAccountBandwidthRate</code>	Modify the Bandwidth consumption multiplier for creating a new account.	Example: <code>[1]</code> . Range: <code>[0, 100.000.000.000.000.000]</code>
#9 — ALLOW.CREATION.OF.CONTRACTS <code>getAllowCreationOfContracts</code>	Enable the creation of new smart contracts.	Example: <code>True</code> Type: <code>True/False</code>
#10 — REMOVE.THE.POWER.OF.THE.GR <code>getRemoveThePowerOfTheGr</code>	Remove the initial voting power held by the Genesis Representatives (GRs).	Example: <code>True</code> Type: <code>True/False</code> - Notice: cannot be set back to <code>False</code> from <code>True</code> .
#11 — ENERGY.FEE <code>getEnergyFee</code>	Modify the fee rate for consuming Energy when an account's Energy is insufficient.	Example: <code>[100]</code> sun per Energy. Range: no explicit value range check in <code>ProposalUtil</code>

Continued on next page

ID / Parameter / API Name	Description	Example / Range
#12 — EXCHANGE.CREATE.FEE getExchangeCreateFee	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: no explicit value range check in ProposalUtil
#13 — MAX.CPU.TIME.OF.ONE.TX getMaxCpuTimeOfOneTx	Modify the maximum CPU execution time allowed for a single transaction.	Example: [80] ms. Range: [10, 400] ms
#14 — ALLOW.UPDATE.ACCOUNT.NAME getAllowUpdateAccountName	Enable the ability for accounts to update their name.	Example: False Type: True/False
#15 — ALLOW.SAME.TOKEN.NAME getAllowSameTokenName	Allow different TRC-10 tokens to be created with the same name.	Example: True Type: True/False
#16 — ALLOW.DELEGATE.RESOURCE getAllowDelegateResource	One-way activation flag for resource delegation. It is a prerequisite for both legacy Stake 1.0 delegation and Stake 2.0 DelegateResourceContract/UnDelegateResourceContract transactions.	Example: True Type: True/False
#18 — ALLOW.TVM.TRANSFER.TRc10 getAllowTvmTransferTrc10	Allow smart contracts to natively transfer TRC-10 tokens using the transferToken call.	Example: True Type: True/False
#19 — TOTAL.CURRENT.ENERGY.LIMIT getTotalEnergyCurrentLimit	Modify the current total amount of Energy available on the network.	Example: [180.000.000.000]. Range: [0, 100.000.000.000.000.000]
#20 — ALLOW.MULTI.SIGN getAllowMultiSign	Enable the multi-signature feature for accounts.	Example: True Type: True/False
#21 — ALLOW.ADAPTIVE.ENERGY getAllowAdaptiveEnergy	Enable the adaptive Energy model.	Example: False Type: True/False
#22 — UPDATE.ACCOUNT.PERMISSION.FEE getUpdateAccountPermissionFee	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 getMultiSignFee	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 getAllowProtoFilterNum	Enable or disable protocol message filtering.	Example: False Type: True/False
#25 — ALLOW.ACCOUNT.STATE.ROOT getAllowAccountStateRoot	Enable recording the account-state root hash in the block header's accountStateRoot field, described in Section 6.1.1.	Example: False Type: True/False
#26 — ALLOW.TVM.CONSTANTINOPLE getAllowTvmConstantinople	Enable the features of the Constantinople Ethereum upgrade in the TVM.	Example: True Type: True/False
#29 — ADAPTIVE.RESOURCE.LIMIT.MULTIPLIER getAdaptiveResourceLimitMultiplier	Modify the multiplier used in the adaptive resource model.	Example: [1000]. Range: [1, 10000]
#30 — ALLOW.CHANGE.DELEGATION getChangeDelegation	Enable the brokerage-based reward-distribution mechanism for SRs and voters, under which block-production rewards and voting rewards for the 27 SRs and 100 SR Partners are accounted for per block.	Example: True Type: True/False
#31 — WITNESS.127.PAY.PER.BLOCK getWitness127PayPerBlock	Modify the voting reward generated for each produced block and shared among the top 127 candidates in proportion to the votes they receive.	Example: [128.000.000] sun - which is 128 TRX. Range: [0, 100.000.000.000.000.000] sun
#32 — ALLOW.TVM.SOLIDITY.059 getAllowTvmSolidity059	Enable TVM support for contracts compiled with Solidity v0.5.9 and above.	Example: True Type: True/False

Continued on next page

ID / Parameter / API Name	Description	Example / Range
#33 — ADAPTIVE_RESOURCE_LIMIT_TARGET_RATIO getAdaptiveResourceLimitTargetRatio	Modify the target ratio used in the adaptive resource model algorithm.	Example: [10]. Range: [1, 1000]
#35 — FORBID_TRANSFER_TO_CONTRACT getForbidTransferToContract	When enabled, reject direct TRX transfers (TransferContract) to any contract account, regardless of whether the contract has a payable or fallback function. Sending TRX as part of a contract call uses TriggerSmartContract and follows the contract's execution rules.	Example: False Type: True/False
#39 — ALLOW_SHIELDED_TRC20_TRANSACTION getAllowShieldedTRC20Transaction	Enable or disable privacy-preserving shielded transactions for TRC-20 tokens.	Example: True Type: True/False
#40 — ALLOW_PBFT getAllowPBFT	Enable the PBFT consensus mechanism for faster transaction finality.	Example: False Type: True/False
#41 — ALLOW_TVM_ISTANBUL getAllowTvmIstanbul	Enable the features of the Istanbul Ethereum upgrade in the TVM.	Example: True Type: True/False
#44 — ALLOW_MARKET_TRANSACTION getAllowMarketTransaction	Enable market order transactions on the decentralized exchange. Since the GreatVoyage-v4.8.1 hard fork, proposals for this parameter are rejected by java-tron; it is not enabled on Mainnet and cannot be activated through a new proposal.	Example: False Type: True/False
#45 — MARKET_SELL_FEE getMarketSellFee	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 getMarketCancelFee	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 getMaxFeeLimit	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 getAllowTransactionFeePool	Enable or disable the transaction fee pool mechanism.	Example: False Type: True/False
#49 — ALLOW_BLACKHOLE_OPTIMIZATION getAllowOptimizeBlackHole	Enable or disable direct burn accounting for applicable network fees instead of transferring them to the blackhole account.	Example: True Type: True/False
#51 — ALLOW_NEW_RESOURCE_MODEL getAllowNewResourceModel	Enable a resource model that separates TRON Power from Bandwidth and Energy and supports staking specifically for TRON Power.	Example: False Type: True/False
#52 — ALLOW_TVM_FREEZE getAllowTvmFreeze	Enable the legacy TVM FREEZE, UNFREEZE, and FREEZEEXPIRETIME instructions defined by TIP-157, allowing smart contracts to perform Stake 1.0-style staking and resource operations.	Example: False Type: True/False
#53 — ALLOW_ACCOUNT_ASSET_OPTIMIZATION getAllowAccountAssetOptimization	Enable an optimization for how account asset (TRC-10) balances are stored.	Example: False Type: True/False
#59 — ALLOW_TVM_VOTE getAllowTvmVote	Enable the ability for smart contracts to execute voting operations.	Example: True Type: True/False
#60 — ALLOW_TVM_COMPATIBLE_EVM getAllowTvmCompatibleEvm	Enable TVM compatibility features with the EVM.	Example: False Type: True/False
#61 — FREE_NET_LIMIT getFreeNetLimit	Modify the amount of free Bandwidth each account receives daily.	Example: [600]. Range: [0, 100000]

Continued on next page

ID / Parameter / API Name	Description	Example / Range
#62 — TOTAL.NET.LIMIT getTotalNetLimit	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 getAllowTvmLondon	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 getAllowHigherLimitForMaxCpuTimeOfOneTx	Allow proposing a higher maximum value for the MAX.CPU.TIME.OF.ONE.TX parameter.	Example: True Type: True/False
#66 — ALLOW.ASSET.OPTIMIZATION getAllowAssetOptimization	Enable account asset storage optimization.	Example: True Type: True/False
#67 — ALLOW.NEW.REWARD getAllowNewReward	Enable the new reward algorithm for SRs and voters.	Example: True Type: True/False
#68 — MEMO.FEE getMemoFee	Modify the fixed fee applied when a transaction contains non-empty memo data.	Example: [1.000.000] sun - which is 1 TRX. Range: [0, 1.000.000.000] sun
#69 — ALLOW.DELEGATE.OPTIMIZATION getAllowDelegateOptimization	Enable the resource-delegation storage optimization. This is a one-way activation flag.	Example: True Type: True/False
#70 — UNFREEZE.DELAY.DAYS getUnfreezeDelayDays	Activate Stake 2.0 and set, in days, the waiting period between initiating an unstake and withdrawing the unstaked TRX.	Example: [14] Days. Range: [1, 365] Days
#71 — ALLOW.OPTIMIZED.RETURN.VALUE.OF.CHAIN.ID getAllowOptimizedReturnValueOfChainId	Enable an optimization for the CHAINID opcode's return value.	Example: True Type: True/False
#72 — ALLOW.DYNAMIC.ENERGY getAllowDynamicEnergy	Enable or disable the dynamic Energy model.	Example: True Type: True/False
#73 — DYNAMIC.ENERGY.THRESHOLD getDynamicEnergyThreshold	Modify the contract Energy consumption threshold before the dynamic penalty applies.	Example: [5.000.000.000]. Range: [0, 100.000.000.000.000.000]
#74 — DYNAMIC.ENERGY.INCREASE.FACTOR getDynamicEnergyIncreaseFactor	Modify the rate at which a contract's dynamic Energy factor increases when its usage exceeds the threshold. The raw value uses a precision factor of 10,000: 2000 represents a 20% increase per cycle.	Example: [2000]. Range: [0, 10000]
#75 — DYNAMIC.ENERGY.MAX.FACTOR getDynamicEnergyMaxFactor	Modify the maximum additional Energy factor under the dynamic Energy model. The raw value uses a precision factor of 10,000: 34000 caps the additional factor at 3.4, for a maximum total multiplier of 4.4 on execution costs to which the factor applies.	Example: [34000]. Range: [0, 100000]
#76 — ALLOW.TVM.SHANGHAI getAllowTvmShangHai	Enable the features of the Shanghai Ethereum upgrade in the TVM.	Example: True Type: True/False
#77 — ALLOW.CANCEL.ALL.UNFREEZE.V2 getAllowCancelAllUnfreezeV2	Enable the ability to cancel all pending unstaking requests under Staking 2.0.	Example: True Type: True/False
#78 — MAX.DELEGATE.LOCK.PERIOD getMaxDelegateLockPeriod	Set the maximum lock.period, in blocks, accepted by Stake 2.0 resource delegation.	Example: [864000]. Range: A new value must be greater than the current value and no greater than 10,512,000 blocks.
#79 — ALLOW.OLD.REWARD.OPT getAllowOldRewardOpt	Enable an optimization for the reward withdrawal algorithm for the old reward system.	Example: True Type: True/False
#81 — ALLOW.ENERGY.ADJUSTMENT getAllowEnergyAdjustment	Enable Energy adjustment functionalities.	Example: True Type: True/False

Continued on next page

ID / Parameter / API Name	Description	Example / Range
#82 — MAX.CREATEACCOUNT.TX.SIZE getMaxCreateAccountTxSize	Modify the maximum transaction size for creating a new account.	Example: [1000]. Range: [500, 10000]
#83 — ALLOW.TVM.CANCUN getAllowTvmCancun	Enable the features of the Cancun Ethereum upgrade in the TVM.	Example: True Type: True/False
#87 — ALLOW.STRICT.MATH getAllowStrictMath	Migrate TVM math operations to <code>java.lang.StrictMath</code> for cross-platform consistency.	Example: True Type: True/False
#88 — CONSENSUS.LOGIC.OPTIMIZATION getConsensusLogicOptimization	Enable optimizations in the consensus logic.	Example: True Type: True/False
#89 — ALLOW.TVM.BLOB getAllowTvmBlob	Enable the TVM compatibility opcodes BLOHASH and BLOBBASEFEE, which currently return zero on TRON.	Example: True Type: True/False
#92 — PROPOSAL.EXPIRE.TIME getProposalExpireTime	Modify the base validity window for governance proposals.	Example: [259,200,000] ms - which is 3 days. Range: (0, 31,536,003,000) ms.
#94 — ALLOW.TVM.SELFDESTRUCT.RESTRICTION getAllowTvmSelfdestructRestriction	Enable EIP-6780-aligned restrictions on SELFDESTRUCT behavior.	Example: True. Type: True/False
Note: Parameters 95–98 were introduced in GreatVoyage-v4.8.2. Parameters 95 and 96 are currently enabled on Mainnet; 97 and 98 remain disabled.		
#95 — ALLOW.TVM.PRAGUE getAllowTvmPrague	Enable governance-gated TVM features aligned with Ethereum Prague/Pectra.	Example: True. Type: True/False
#96 — ALLOW.TVM.OSAKA getAllowTvmOsaka	One-way activation of the supported TVM features aligned with Ethereum Osaka. The proposal value must be 1; once enabled, the feature cannot be disabled through governance.	Activation value: True (1) Behavior: One-way
#97 — ALLOW.HARDEN.RESOURCE.CALCULATION getAllowHardenResourceCalculation	Enable hardened resource-calculation behavior introduced with v4.8.2.	Example: False. Type: True/False
#98 — ALLOW.HARDEN.EXCHANGE.CALCULATION getAllowHardenExchangeCalculation	Enable hardened exchange-calculation behavior introduced with v4.8.2.	Example: False. Type: True/False