TronFlash Protocol

First Production-Grade Flashloan Infrastructure for TRON

Technical Architecture & Critical Specification Document

Version 1.0 | January 2026

Classification: Technical Design Specification

Document Type	Architecture Specification
Target Chain	TRON Mainnet / Nile Testnet
Protocol Category	DeFi Infrastructure — Flashloan Provider
Estimated TVL Target	$50M — $500M (Year 1)

1. Executive Summary

1.1 Strategic Opportunity

TRON represents the second-largest DeFi ecosystem by TVL ($8.25B) with no production-grade flashloan infrastructure. This document specifies the technical architecture for TronFlash, the first native flashloan protocol on TRON, enabling zero-capital MEV extraction, arbitrage, liquidations, and collateral swaps.

1.2 Market Gap Analysis

Metric	TRON Current State	Opportunity
Total DeFi TVL	$8.25B	Second only to Ethereum
Flashloan Protocols	0 (production)	First-mover advantage
JustLend TVL	$5.95B	Massive liquidation market
MEV Competition	Near zero	Blue ocean extraction
Q2 2025 MEV Revenue	$165M (chain-wide)	Capturable with infrastructure

1.3 Protocol Value Proposition

- Enable zero-capital DeFi strategies on TRON (arbitrage, liquidations, collateral swaps)
- Generate protocol revenue through flashloan fees (0.05-0.09% per loan)
- Bootstrap liquidity through competitive yield for depositors
- Capture strategic MEV extraction capabilities as protocol operator

2. TRON Technical Foundation

2.1 TRON Virtual Machine (TVM) Architecture

TVM is fundamentally compatible with EVM, enabling direct Solidity contract deployment with minor adaptations. Key architectural differences affect flashloan implementation:

Characteristic	Ethereum/EVM	TRON/TVM
Resource Model	Gas (variable pricing)	Energy + Bandwidth (fixed pricing)
Execution Cost	gasPrice * gasUsed	energyPrice * energyUsed (100 sun)
Block Time	~12 seconds	~3 seconds
TPS Capacity	~15-30 TPS	~2,000 TPS
Consensus	PoS (validators)	DPoS (27 Super Representatives)
Solidity Support	Native	Full compatibility
Contract Callbacks	Unlimited depth	Supported (no explicit limit)

2.2 Energy & Bandwidth Resource Model

TRON's dual-resource model affects transaction cost economics. Flashloan users must account for both resources:

Energy (Smart Contract Execution)

- Consumed by all smart contract operations (computation, storage writes)
- energyPrice: 420 sun (current mainnet parameter)
- Acquisition: Freeze TRX (staking) or burn TRX at execution
- Typical flashloan + swap: 200,000-500,000 energy (~$0.50-2.00)

Bandwidth (Transaction Data)

- Consumed by transaction data size (calldata, signatures)
- Free allocation: 5,000 bandwidth points/day per account
- Excess consumption: Burn TRX at 1,000 sun per bandwidth point
- Typical flashloan transaction: 300-600 bandwidth points

2.3 TRC-20 Token Standard

TRC-20 mirrors ERC-20 exactly, enabling direct interface compatibility. Flashloan protocol will support all TRC-20 tokens with sufficient liquidity:

Token	Contract Address	Liquidity Priority
USDT	`TR7NHqjeKQxGTCi8q8ZY4pL8otSzgjLj6t`	Critical ($48B+ circulation)
USDC	`TEkxiTehnzSmSe2XqrBj4w32RUN966rdz8`	High
USDD	`TPYmHEhy5n8TCEfYGqW2rPxsghSfzghPDn`	High (TRON native stable)
WTRX	`TNUC9Qb1rRpS5CbWLmNMxXBjyFoydXjWFR`	High

BTT	`TAFjULxiVgT4qWk6UZwjqwZXTSaGaqnVp4`	Medium

3. Core Protocol Architecture

3.1 System Overview

TronFlash implements a pool-based flashloan architecture inspired by Aave V3, adapted for TVM's resource model. The system comprises four primary smart contract components:

Contract	Responsibility
FlashLoanPool	Core liquidity pool; manages deposits, withdrawals, and flashloan execution
FlashLoanReceiver	Interface contract that borrowers must implement; defines executeOperation callback
PoolRegistry	Multi-token pool management; maps tokens to pool addresses; handles pool creation
FeeCollector	Protocol fee accumulation; handles fee distribution to treasury and depositors

3.2 Execution Flow Specification

The flashloan execution follows a strict atomic sequence. All operations must complete within a single transaction or the entire transaction reverts:

Phase 1: Loan Initiation

1. Borrower contract calls FlashLoanPool.flashLoan(token, amount, receiverAddress, params)
2. Pool validates: sufficient liquidity, receiver implements IFlashLoanReceiver interface
3. Pool calculates premium: premium = amount * flashLoanFee / 10000 (default: 5 bps = 0.05%)

Phase 2: Fund Transfer

4. Pool transfers requested amount to receiver contract via TRC-20 transfer()
5. Pool records pre-execution balance for verification

Phase 3: Callback Execution

6. Pool calls receiver.executeOperation(token, amount, premium, initiator, params)
7. Receiver executes arbitrary logic (arbitrage, liquidation, collateral swap, etc.)
8. Receiver must approve Pool for amount + premium before returning

Phase 4: Repayment Verification

9. Pool pulls amount + premium from receiver via transferFrom()
10. Pool verifies post-execution balance >= pre-execution balance + premium
11. If verification fails: entire transaction reverts (atomic guarantee)
12. Premium distributed: treasury share + depositor yield accrual

4. Smart Contract Specification

4.1 IFlashLoanReceiver Interface

All borrower contracts must implement this interface to receive flashloans:

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.20;

interface IFlashLoanReceiver {
    function executeOperation(
        address token,
        uint256 amount,
        uint256 premium,
        address initiator,
        bytes calldata params
    ) external returns (bool);
}
```

Parameter Specification

Parameter	Type	Description
`token`	`address`	TRC-20 token address being borrowed
`amount`	`uint256`	Amount of tokens borrowed (in token decimals)
`premium`	`uint256`	Fee owed in addition to amount (amount * fee / 10000)
`initiator`	`address`	Address that initiated the flashloan (msg.sender to pool)
`params`	`bytes`	Arbitrary data passed through from flashLoan() call

4.2 FlashLoanPool Core Functions

flashLoan() — Primary Entry Point

```
function flashLoan(
    address receiverAddress,
    address token,
    uint256 amount,
    bytes calldata params
) external nonReentrant returns (bool);
```

deposit() — Liquidity Provider Entry

```
function deposit(
    address token,
    uint256 amount
) external returns (uint256 shares);
```

Depositors receive pool shares (ERC-4626 vault pattern) representing their proportion of the pool. Shares accrue value as flashloan fees accumulate.

withdraw() — Liquidity Provider Exit

```
function withdraw(
    address token,
    uint256 shares
) external returns (uint256 amount);
```

5. Security Model & Risk Analysis

5.1 Attack Vector Analysis

Attack Vector	Description	Mitigation
Reentrancy	Attacker re-enters flashLoan during callback	nonReentrant modifier; CEI pattern
Flash Loan Chaining	Taking flashloan to manipulate pool state	Balance verification post-callback
Oracle Manipulation	Using flashloan to manipulate price oracles	Not protocol's responsibility; user risk
Griefing Attack	Depositing then using flashloan to steal fees	Fee accrual occurs after successful repayment
Malicious Token	Token with transfer hooks causing reentrancy	Token whitelist; nonReentrant on all entries
Integer Overflow	Overflow in premium calculation	Solidity 0.8+ native overflow checks

5.2 Critical Security Requirements

Reentrancy Protection

All external entry points (flashLoan, deposit, withdraw) must implement OpenZeppelin's ReentrancyGuard. The nonReentrant modifier prevents recursive calls that could exploit intermediate states.

Checks-Effects-Interactions Pattern

13. CHECKS: Validate all inputs (amount > 0, sufficient liquidity, valid receiver)
14. EFFECTS: Update internal state (record loan in progress, pre-balance snapshot)
15. INTERACTIONS: External calls last (transfer, callback, transferFrom)

Token Whitelist Strategy

Initial deployment supports only verified TRC-20 tokens (USDT, USDC, USDD, WTRX). Token whitelist prevents malicious token contracts with transfer hooks or non-standard behavior. Governance can add tokens after security review.

5.3 Audit Requirements

- Pre-deployment: Two independent security audits (e.g., Trail of Bits, OpenZeppelin)
- Formal verification: Certora or Echidna property-based testing
- Bug bounty: $50K-250K program via Immunefi post-launch
- Ongoing: Weekly security monitoring; emergency pause capability

6. Economic Model & Fee Structure

6.1 Fee Architecture

Fee Type	Rate	Recipient
Flashloan Premium	5 bps (0.05%)	Split: 80% depositors, 20% treasury
Deposit Fee	0 bps	No fee to encourage liquidity
Withdrawal Fee	0-10 bps	Optional; can be used to discourage bank runs
Protocol Fee (Treasury)	20% of premium	Protocol treasury; governance controlled

6.2 Revenue Projections

Revenue model based on flashloan volume assumptions. TRON's Q2 2025 MEV revenue of $165M provides baseline for demand estimation:

Scenario	Daily Volume	Annual Volume	Protocol Revenue
Conservative	$10M	$3.65B	$365K (20% of 0.05%)
Base Case	$50M	$18.25B	$1.825M
Optimistic	$200M	$73B	$7.3M

Depositor Yield Calculation:

With $50M TVL and $50M daily flashloan volume (100% utilization), depositors earn: ($50M * 0.05% * 80%) / $50M = 0.04% daily = ~14.6% APY. At 50% utilization: ~7.3% APY.

6.3 Liquidity Bootstrapping Strategy

16. Phase 1 (Launch): Seed liquidity from protocol treasury ($500K-1M)
17. Phase 2 (Incentive): Token rewards for early depositors (if token launched)
18. Phase 3 (Integration): Partner with JustLend for collateral utilization
19. Phase 4 (Scale): Institutional LP onboarding; yield optimization vaults

7. Integration Architecture

7.1 JustLend Liquidation Integration

Primary use case: flashloan-powered liquidations on JustLend ($5.95B TVL). Integration requires calling JustLend's liquidation function within the flashloan callback:

Execution Flow

20. Flashloan USDT (debt token) from TronFlash
21. Approve JustLend jToken contract for USDT
22. Call jToken.liquidateBorrow(borrower, repayAmount, jTokenCollateral)
23. Receive seized collateral (jTokens)
24. Redeem jTokens for underlying collateral
25. Swap collateral to USDT via SunSwap
26. Approve TronFlash for USDT (amount + premium)
27. Return true from executeOperation(); profit remains in contract

7.2 SunSwap Arbitrage Integration

Arbitrage between SunSwap pools or against external venues. Integration uses SunSwap's router for atomic swaps within the flashloan callback:

Key Contracts

Contract	Address (Mainnet)	Purpose
SunSwap Router V2	`TKzxdSv2FZKQrEqkKVgp5DcwEXBEKMg2Ax`	Swap execution
SunSwap Factory V2	`TXk8rQSAvPvBBNtqSoY6nCfsXWCSSpTVQF`	Pair discovery
JustLend Comptroller	`TGjYzgCyPobsNS9n6WcbdLVR9dH7mWqFx7`	Market data

8. Deployment Roadmap

8.1 Development Phases

Phase	Timeline	Deliverables
1. Foundation	Weeks 1-3	Core contracts (FlashLoanPool, interfaces); TronBox setup; unit tests
2. Integration	Weeks 4-6	JustLend adapter; SunSwap adapter; end-to-end integration tests
3. Security	Weeks 7-10	Internal audit; external audit engagement; bug fixes; formal verification
4. Testnet	Weeks 11-12	Nile testnet deployment; public testing; documentation
5. Mainnet	Week 13+	Mainnet deployment; seed liquidity; monitoring infrastructure

8.2 Development Environment

Required Tools

- TronBox 3.3+: Smart contract compilation, deployment, testing
- TronWeb 5.2+: JavaScript library for contract interaction
- Solidity 0.8.20: Compiler version (native overflow protection)
- OpenZeppelin Contracts 5.0: ReentrancyGuard, Ownable, ERC4626
- TronLink Wallet: Deployment and testing

Network Configuration

Network	Full Node RPC	Purpose
Nile Testnet	`https://nile.trongrid.io`	Development and testing
Shasta Testnet	`https://api.shasta.trongrid.io`	Secondary testing
Mainnet	`https://api.trongrid.io`	Production deployment

9. Risk Assessment & Feasibility

9.1 Technical Risk Matrix

Risk Factor	Likelihood	Impact	Mitigation Strategy
Smart Contract Bug	Medium	Critical	Multiple audits; formal verification; bug bounty
Low Liquidity	High	Medium	Yield incentives; institutional partnerships; seed capital
Competition	Low	Medium	First-mover advantage; rapid deployment
TRON Network Issues	Low	High	Monitor SR performance; emergency pause
Regulatory Action	Low	Medium	Decentralized governance; no user funds custody
Oracle Dependency	N/A	N/A	No oracle required for core flashloan functionality

9.2 Feasibility Assessment

Technical Feasibility: HIGH

- TVM is fully compatible with EVM; Aave-style flashloan architecture directly portable
- TronBox/TronWeb tooling mature and well-documented
- No architectural blockers identified; callback pattern supported

Market Feasibility: HIGH

- Clear demand signal: $8.25B TVL with zero flashloan infrastructure
- JustLend liquidation market untapped; no competing liquidators
- First-mover advantage in blue ocean market

Operational Feasibility: MODERATE-HIGH

- Liquidity bootstrapping is primary challenge; requires seed capital or incentives
- Security audit costs: $50-150K for comprehensive coverage
- Ongoing operational costs: monitoring, upgrades, support

10. Conclusion & Recommendations

10.1 Strategic Recommendation

TronFlash represents a high-confidence, high-return opportunity. The combination of massive addressable market ($8.25B TVL), zero competition, and proven technical architecture (Aave V3 pattern) creates favorable conditions for first-mover capture of TRON's flashloan market.

10.2 Recommended Next Steps

28. Immediate: Begin core contract development using TronBox; target 3-week foundation phase
29. Week 4: Engage security auditor (Trail of Bits, OpenZeppelin, or equivalent)

30. Week 8: Deploy to Nile testnet; begin public testing campaign
31. Week 12: Mainnet deployment with $500K-1M seed liquidity
32. Post-launch: Deploy own MEV extraction infrastructure using TronFlash

10.3 Success Criteria

Metric	6-Month Target	12-Month Target
TVL	$10M	$50M
Daily Flashloan Volume	$5M	$50M
Protocol Revenue (Annual)	$100K	$1M+
Active Integrations	5+ projects	20+ projects

— END OF SPECIFICATION —

Document Version 1.0 | Prepared January 2026