OP_NET: The Bitcoin-Aligned Smart Contract Metaprotocol (*Litepaper*)

Introduction

Bitcoin is renowned for its security, decentralization, and provenance. However, its scripting language, Bitcoin Script, does not directly allow for advanced smart contracts and applications. This gap has driven numerous efforts to extend Bitcoin's capabilities, from Colored Coins and Counterparty early on, to Ordinals and BRC-20 [meta] meta-protocols most recently. However, each of these has faced limitations in functionality, scalability, and user experience.

The limitation of Ordinals, BRC-20 and Runes is well-documented. For example, to exchange assets, marketplaces must utilize Partially Signed Bitcoin Transactions (PSBTs) to remain non-custodial. In practice, this has created an NFT-like trading environment for all tokens. There are also no decentralized exchanges (DEXs) for BRC-20 or Runes, despite numerous attempts, because the indexers are restricted to only a few commands: Deploy, Mint and Transfer. Multiple "DEXs" have been announced since BRC-20's creation, and some have launched, but they require users to effectively custody their tokens with a centralized platform, whether via overtly depositing tokens or through the use of multi-signature wallets controlling the assets.

OP_NET is a new metaprotocol operating outside the Ordinals envelope that is purpose-built to address this limitation, integrating seamlessly with Bitcoin while introducing a comprehensive suite of smart contract capabilities via Tapscript. At its core, OP_NET functions to transform the programmability of the Bitcoin blockchain into the programmability of smart contract blockchains, without any BIPs.

OP_NET not only addresses the limitations of previous metaprotocols but also provides a robust infrastructure for DeFi applications, wrapped Bitcoin, NFTs, and more. By using native Bitcoin for transaction fees and indexers that compile in WebAssembly (Wasm) for multi-language compatibility, OP_NET provides a more seamless user and developer experience.

The ideal Bitcoin-aligned metaprotocol

In designing the ideal metaprotocol within the constraints of Script, we have deduced certain properties we believe it should possess, learning from our predecessors:

1. Turing completeness with the full suite of smart contract functionality that allows for DeFi on Bitcoin, including the ability to permissionlessly deploy smart contracts
2. Bitcoin as the data availability, consensus, and settlement layer, minimizing trust assumptions
3. Immutable indexation with minimal latency and mechanisms to prevent and mitigate manual state alterations
4. Bitcoin as the gas token, to remain uncompromisingly Bitcoin-aligned
5. Mechanisms to exchange Bitcoin for assets in a trustless or trust-minimized manner
6. Seamless UX (e.g., single-address system) and DevEx (e.g., out-of-the-box developer packages and the ability to write in multiple languages)
7. No Bitcoin Improvement Proposal (BIP) required, but upgradeable for when changes in Bitcoin Core are implemented

These principles are the foundations upon which OP_NET has been built. We cover its key features below.

OP_NET: Key differentiating features

Bitcoin Layer 1 mechanisms

As with all Bitcoin meta-protocols, OP_NET retains Bitcoin block consensus and transaction data availability on Bitcoin Layer 1, with an execution VM that performs complex computation based on arbitrary data published on blocks. The diagram below illustrates the role of OP_NET as a smart contract execution layer for Bitcoin (Figure 1).

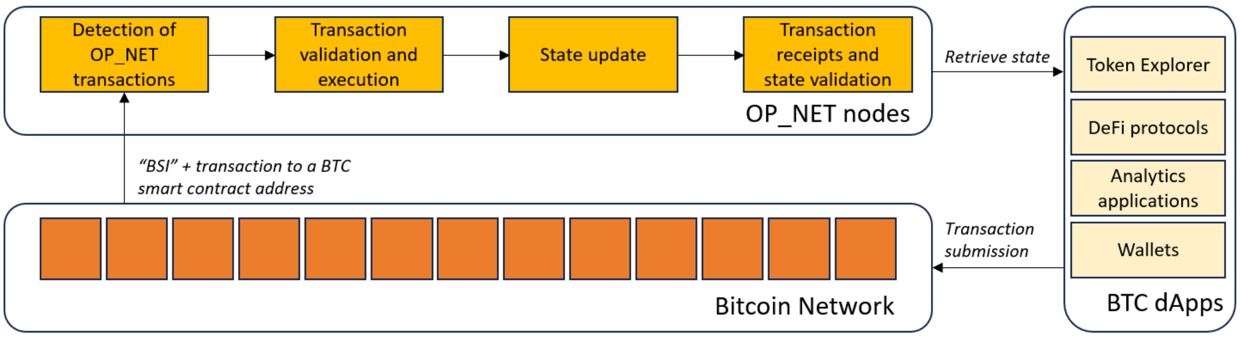

Figure 1: Bitcoin and OP_NET overview

All OP_NET transactions submitted to the Bitcoin network are labeled in an arbitrary data field with the string "BSI," indicating that an OP_NET transaction is initiated; these transactions are directed toward unspendable Bitcoin addresses that have been abstracted into contract addresses. OP_NET will detect these transactions and execute them in the OP_NET virtual machine (OP_VM), updating contracts state accordingly.

Each transaction is processed to determine which storage slots are affected. The new state of each affected storage slot is computed based on the transaction's calldata. Merkle trees are employed to generate cryptographic proofs that verify the correctness of the storage state. For every executed transaction, a transaction receipt is generated providing details about the transaction, including the changes made to storage slots and any events emitted. A state root is also generated to prove that the states changed by the transaction were not altered in any way. The receipts and states of every transaction are summarized in a checksum root, which is included in the block header for verification.

Applications and entities running their own OP_NET nodes can easily retrieve token state via API, which we provide through a variety of NPM packages and through an OP_NET token explorer.

On-chain smart contract addresses, deployment and interactions

Bitcoin does not inherently support hosting contract addresses on-chain, as each address is traditionally tied to a private key that cannot be renounced. However, OP_NET abstracts Bitcoin addresses into interactable contract addresses using a specific type of Bitcoin transaction known as a Taproot script-path spend, which allows for advanced spending conditions. This unlocks:

1. Deployment of OP_NET smart contracts under an on-chain Bitcoin address that is not under the dominion of a user's wallet
2. User interaction on-chain with designated Bitcoin addresses housing smart contracts

Practically, what this means is that all smart contract address, deployment and interaction data remain fully on-chain for Bitcoin users to view as transactions are submitted (Figure 2).

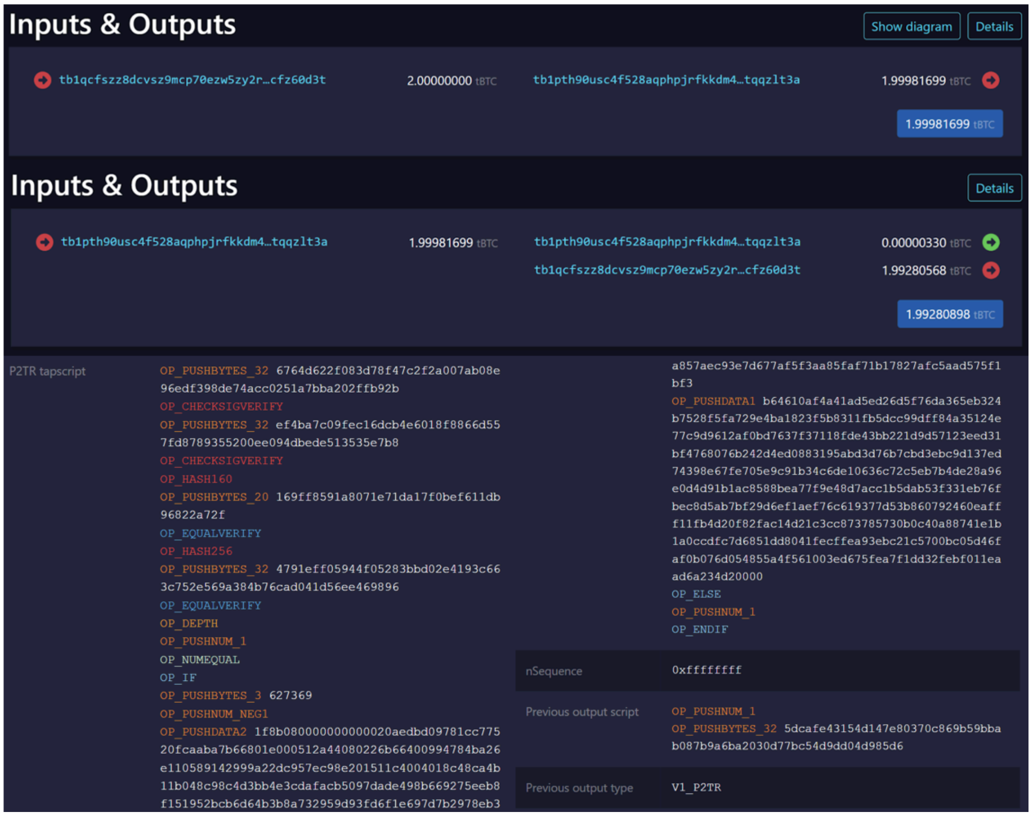

Figure 2: Example contract deployment script in Bitcoin testnet

In this example, the address tb1q...fz60d3t employs a script-path spend to deploy a token contract, which programmatically generates a Taproot address tb1p...qqzlt3a and an associated SegWit address. These addresses act as unspendable contract addresses for OP_NET. To ensure transactions are not considered "dust" and excluded from block construction, the contract deployer or interactor must spend at least 330 satoshis (sats) into the contract address with each transaction. These 330 sats are unspendable and effectively burned.

The deployment script includes two signature checks using the OP_CHECKSIGVERIFY opcode, verifying the signatures of both the user deploying the contract and the contract/interaction keypair itself. This ensures that all deployment and interaction transactions include these signatures for authenticity and security. Additionally, the smart contract's deployment bytecode is submitted on-chain during the deployment process using the OP_PUSHDATA opcode, making the bytecode visible to all users and maintaining transparency.

There are no inherent size limitations for smart contracts on OP_NET, with the maximum deployable contract size being 4 MB, the limitation for a Bitcoin block. To reduce deployment and interaction fees, the bytecode within the transaction is compressed using ZLIB. This approach ensures secure, transparent, and efficient deployment and interaction with smart contracts on OP_NET.

Single-address UX compatible with all Bitcoin address types

Unlike Runes and Ordinals, OP_NET contracts are not tied to specific UTXOs. Instead, they are virtualized within the OP_NET VM (OP_VM), which simplifies token management and significantly reduces the risk of accidental spending. In OP_NET, transactions are initiated by spending at least 330 satoshis (sats) into the contract address associated with a contract deployment. This design ensures that spending Bitcoin from a user's wallet does not inadvertently spend their OP_NET tokens, eliminating such risks.

The system allows for a streamlined, one-address approach compatible with Taproot and SegWit Bitcoin addresses. Users can continue to use their existing Ordinals or Runes wallets as they normally would, without any modifications. This integration provides seamless interaction between Bitcoin and OP_NET, maintaining security and ease of use.

OP_VM, a WebAssembly (Wasm) state machine, as the execution layer

OP_NET nodes are specialized nodes responsible for executing smart contract code and managing interactions via the OP_VM. They are explicitly designed for high throughput execution, offering several key improvements over current metaprotocol indexers:

1. *Minimal latency and scalable performance through multithreading:* By isolating each part of the software into separate threads, these nodes handle multiple tasks concurrently, avoiding execution bottlenecks and enhancing responsiveness. This design also allows for dynamic load balancing to maximize throughput.

2. *Tamper-resistant and secure execution:* Nodes employ Merkle trees and storage proofs to ensure data integrity and security. Contract execution is isolated using Rust and Wasmer, protecting against malicious attacks and ensuring correct contract logic execution.

3. *Support Bitcoin block reorganizations and recovery:* They are equipped with features to actively manage reorgs, re-executing transactions as necessary to maintain correct state.

4. *Efficient data management:* Indexers use ZLIB compression for bytecode, translating to smaller transaction sizes and lower fees. Data compression also decreases storage requirements for nodes

OP_NET nodes function as read-only local state machines, executing transactions similarly to Ordinals but within OP_NET's unique framework. They compute and record storage slot state changes with each block, using Merkle trees to generate and validate storage proofs. This process ensures data integrity and security, with mechanisms in place to revert state changes if block reorganization or malicious action is detected.

Smart contracts on OP_NET are compiled in WebAssembly (Wasm), making them compatible with over 20 programming languages, including AssemblyScript, Rust, Python, C/C++, and Go. Currently, there is out-of-the-box VM support for AssemblyScript, with plans to create packages supporting other languages. We plan to support Rust next.

To ensure deterministic execution and mitigate the risk of infinite loops, OP_VM employs computationally-based gas, tracking each operation at the VM level. This gas, paid for with Bitcoin and subsequently burned, ensures that computational resources are appropriately accounted for. Additionally, OP_VM limits re-entrancy by default, allowing contracts to be called only once in contract-to-contract interactions. If this condition is not met, the transaction is reverted by the VM. However, users have the option to disable this restriction for their contracts if required.

Bitcoin as the one and only gas token

To utilize OP_NET, only native Bitcoin is required for transaction fees, obviating the need for a separate token for node incentivization or fee payments. OP_NET does not have a bespoke gas token and OP_NET’s Bitcoin fees over a specified threshold will be paid to the indexer network (the initial threshold is set at 250,000 satoshi or 0.0025 BTC).

Bitcoin uses topological transaction ordering (TTOR) to determine the prioritization of transactions within a block: Transactions can appear in any order and miners are not obligated to sort them by fees. This is vulnerable to miner extractable value (MEV) and we partially mitigate this through an on-chain priority fee structure, as shown below. Note that the OP_NET fees below are in Bitcoin, but are incremental to the baseline Bitcoin fee rates. These fees can be reflected in-wallet or front-end UI.

Total BTC Transaction Fee = BTC Network Base Fee + OP_NET Transaction Fee
OP_NET Transaction Fee = Execution Fee + Priority Fee

- BTC Network Base Fee - the fee required to be included *on* a Bitcoin block
- OP_NET Execution Fee - the gas fee required to perform the desired contract deployment or interaction
- OP_NET Priority Fee - an optional, additional fee to obtain higher execution priority *within* a Bitcoin block

The minimum OP_NET Transaction fee is 330 sats (the cost to interact with a contract address via Taproot script-path spend); this is the minimum BTC value that is not considered dust by the network. Transactions that confirm within the block are ordered by OP_NET according to the OP_NET Transaction Fee, in descending order. By prioritizing users who burn the most Bitcoin, OP_NET simultaneously limits MEV.

If OP_NET Transaction Fee ≥ OP_NET Fee Threshold (0.0025 BTC), 330 sats will be burned into the contract address, and the remainder of the fee will be paid to the node network. Conversely, if OP_NET Transaction Fee < OP_NET Fee Threshold, the entirety of the OP_NET Transaction Fee will be burned into the contract address. This simplifies UTXO management for OP_NET nodes while still rewarding them for maintaining the network.

Multiple ways to onboard, including smart contract-compatible Wrapped Bitcoin via the node network

OP_NET offers multiple ways for users to onboard into its token ecosystem:

1. Minting tokens with native BTC
2. Purchasing tokens directly with native BTC using partially-signed Bitcoin transactions (PSBTs)
3. Wrapping BTC into OP_NET-compatible standards using a Proof of Authority (PoA) system and swapping into tokens on OP_NET
4. Acquiring tokens from centralized exchanges

To fully utilize BTC within OP_NET's contracts, it must be converted into OP_NET-compatible Wrapped BTC (WBTC), analogous to Wrapped Ether (WETH) on Ethereum. This conversion allows BTC to be used in a variety of DeFi applications, such as swaps and lending. Because of Bitcoin Script, it is not feasible to automatically wrap BTC for compatibility in the same way as ETH and there is currently no precedent for on-chain WBTC. To address this, we introduce an on-chain Proof of Authority (PoA) vaults mechanism that utilizes multi-signature leaf scripts overseen by trusted OP_NET nodes in an opt-in system. The vaults system is designed to optimize BTC UTXO management and WBTC minting/burning when wrapping and unwrapping. All data transmitted between trusted indexers of the WBTC vaults system is encrypted using post-quantum cryptographic techniques (IETF variant of ChaCha20-Poly1305 encrypted, ed25519 signed encryption). This wrapping mechanism can be used for assets beyond Bitcoin, but we have designed strictly for Bitcoin on initial release.

Trusted indexers earn from fees associated with wrapping and unwrapping BTC in this system. They initially cover transaction fees, then deduct these transaction fees along with a minor service fee from the BTC amount returned to the user, ensuring they are compensated for their expenses. This service fee is set globally at the protocol level and is not determined at the local level by indexers. A portion of this service fee is allocated toward nodes and a portion toward WBTC stakers, incentivizing network TVL.

This does not preclude other OP_NET-compatible BTC solutions that are devised by other developers.

Fungible and non-fungible token standards, OP_20 and OP_721

OP_NET extends Bitcoin's functionality to support fungible and non-fungible tokens (OP_20 and OP_721), analogous to ERC-20 and ERC-721 on Ethereum. As the name suggests, OP_20 will be the primary fungible standard and OP_721 the primary non-fungible standard for OP_NET.

Out-of-the-box developer packages and OP_NET token explorer

OP_NET is equipped with a token explorer, a JSON-RPC architecture similar to Go-Ethereum (GETH) and a multitude of NPM packages akin to the "ethers" web3 library. These tools facilitate the construction of smart contracts and applications on Bitcoin, enabling developers to create and deploy sophisticated dApps with ease. This extensive developer support ensures that OP_NET is highly integrable with existing DeFi applications (e.g., analytics and token charting websites).

Compatibility with Bitcoin upgrades

OP_NET can be easily upgraded in the future with new BIPs, for example OP_CAT, if identified as beneficial to the protocol. Indexers are upgradable via consensus upgrades using PoA.

Individual sections detailing each feature of OP_NET will be released in greater detail with the OP_NET White Paper.