OP_NET Consensus and Execution Protocol Specification

Introduction and Design Goals

OP_NET is a Bitcoin Layer-1 metaprotocol enabling complex smart contracts on Bitcoin without sidechains or custodial trust. It extends Bitcoin's functionality by using Bitcoin transactions as the delivery mechanism for **OP_VM** contract calls, with native BTC as the gas/fee currency. All contract data is embedded or anchored in Bitcoin's blockchain, avoiding any external data feeds or multi-signature coordination. The core design goals are to achieve **provable security** (every state change is verifiable from on-chain data and cryptographic proofs), **full decentralization** (no privileged sequencers or stakers – any node can participate in validation), and seamless atomic interaction between Bitcoin UTXOs and OP_NET's internal state. The protocol is implemented in two phases – a straightforward **Pre-PoCtm phase** using on-chain transactions for every contract call, and a more advanced **Post-PoCtm phase** that introduces ephemeral sub-blocks with a novel ***Proof of Continuum*** consensus model. The system remains modular and upgradeable, meaning new cryptographic primitives or VM improvements can be integrated over time without fundamental redesign.

Key Objectives:

- **Native Bitcoin Integration:** Use Bitcoin's L1 blocks for transaction ordering and gas payment, **embedding smart contract calls in Bitcoin transactions**. Avoid introducing any separate tokens or sidechains – BTC is the sole asset for fees and settlement.
- **Provable and Atomic Execution:** Ensure that whenever a Bitcoin UTXO is consumed or created as part of a contract (e.g. swapping BTC for an OP_20 token), the corresponding OP_NET state change occurs atomically and verifiably in the same operation. If an on-chain transfer is reversed (e.g. via reorg), the state change is reverted in lockstep.
- **Permissionless Consensus:** Block producers with a combination of deterministic ordering rules and cryptographic proofs. Any honest node can independently recompute and verify the state (via **Proof of Calculation**), and no identity or reputation is needed for participation. **Proof of Validation** by other nodes serves only as a sanity check for discrepancies – it does not rely on any voting or staking, preserving Bitcoin's trustless model.
- **Continuity and Finality:** Leverage Bitcoin's proof-of-work finality for overall security, while introducing a *continuum* of faster sub-blocks regulated by a Verifiable Delay Function (VDF). This provides predictable, **time-sequenced slots** for contract execution between Bitcoin blocks, without allowing malicious actors to manipulate ordering or rush ahead. All state transitions remain cryptographically verifiable to prevent tampering.
- **Modularity and Future-Proofing:** Architect the system in layers (transaction format, execution VM, consensus logic, data commitments) such that each layer can be upgraded. For example, the VM (a WebAssembly-based **OP_VM**) can support new

languages or opcodes without affecting consensus. Cryptographic components (hash functions, VDF algorithm, proof systems) can be swapped or improved as needed. This ensures the protocol can evolve (e.g. add zk-SNARK proofs for state transitions, or adjust gas rules) while maintaining backward compatibility where possible.

Cryptographic Primitives and Data Structures

OP_NET employs well-established cryptographic primitives, inherited from or compatible with Bitcoin, to guarantee security and verifiability:

- **Hash Functions and Merkle Trees:** All OP_NET data commitments use SHA-256 hashing to form Merkle trees. Each ephemeral block of transactions is summarized by a **Merkle root** hash. This root is published on-chain and serves as a commitment to the entire set of smart-contract transactions in that batch. Given the root, any node can verify a transaction's inclusion via a Merkle **inclusion proof**, which consists of the transaction's hash and a path of sibling hashes that reconstruct the root. Merkle trees allow OP_NET users to audit that their transaction was included and executed correctly. Additionally, the global OP_NET *state* (contract storage, account balances, etc.) can be hashed into a state Merkle root for quick comparison between nodes (the design allows but does not *require* state roots on-chain; see *State Management* below).

- **Digital Signatures:** OP_NET relies on Bitcoin's native signature schemes (ECDSA/Schnorr on secp256k1) for authenticating ownership of UTXOs and user actions. Every Bitcoin UTXO referenced in a contract call must be unlocked by a valid signature from its owner, just like any Bitcoin transaction. These signatures are verified by all OP_NET nodes when processing the transaction (and by Bitcoin miners when including the transaction). OP_NET's **Unified Account** model means a single Schnorr keypair can control both on-chain BTC and contract assets, simplifying authentication. Moreover, OP_NET's VM can verify Schnorr signatures within smart contracts if needed (e.g. to authorize a delegated operation) since contract code can access the same cryptography (this is enabled by Taproot's capabilities).

- **Verifiable Delay Function (VDF):** To enforce time-based ordering in the Post-PoCtm phase, OP_NET uses a Verifiable Delay Function. A VDF is a cryptographic function that *requires* a preset number of sequential steps to compute, with no shortcut via parallelism, and yields a proof that can be verified quickly. OP_NET's consensus uses a VDF to impose a minimum real-time delay between ephemeral blocks. The VDF's output serves as an unpredictable but deterministic seed for certain ordering decisions (like tie-breakers) and as proof that the required time has passed for the next slot. Each node computes the VDF, and all honest nodes will arrive at roughly the same result around the same time. The attached proof lets any node verify the result near-instantly without redoing the entire computation. This ensures **Proof of Continuum** – the system's timeline advances uniformly for everyone, and no one can accelerate the clock or produce sub-blocks faster than the protocol permits.

- **Proof of Calculation (PoC):** This refers to a **cryptographic receipt of execution** that a node provides after computing a batch of transactions. In OP_NET, the node (or nodes) that execute a given ephemeral block will output evidence that the computations were performed correctly. One simple form of PoC is the Merkle root of all executed transactions plus a hash of the final state – by publishing these, a node commits to a specific outcome that others can verify by re-execution. In future or advanced implementations, PoC could be a zk-SNARK or STARK proving the validity of the state transition. The goal is that a **single node's heavy computation can be independently verified by others with much less work**. As an analogy, "the node that did the work generated a cryptographic receipt showing it actually performed [the computation]" – that receipt (like a validity proof) is broadcast along with the new state root. All other nodes check the proof or recompute the transactions to confirm no errors. PoC ensures no central actor can cheat in executing contracts; even if only one node computed the block, everyone else can validate the results against the published cryptographic commitments.

- **Proof of Validation (PoV):** In addition to the automated proofs, OP_NET encourages a subset of nodes (so-called *bootstrap nodes*) to actively validate each state transition and flag any discrepancies. These nodes are not required for consensus – they are "informal" in that they don't sign blocks in a ledger – but they serve as an early warning system. If a PoC proof were missing or incorrect or if a malicious anchor transaction somehow included an invalid state root, the PoV nodes would detect the mismatch by recomputing the state and could alert the network (or reject that anchor). In normal operation, PoV is simply every node verifying everything; the term bootstrap nodes refers to well-connected nodes that might provide fast validation or checkpoints to new users. The system does **not** rely on a quorum or reputation of validators to finalize state, it merely uses their independent verification as an added safety net. The only authoritative source of ordering is the Bitcoin blockchain and the protocol rules – *not* any vote by validators. This keeps OP_NET's security model aligned with Bitcoin's (no new trust assumptions).

- **Deterministic Randomness (for Tie-breaking):** In cases where protocol rules require a random choice (e.g. tie-break between two transactions with identical gas and fee), OP_NET derives randomness from verifiable sources like VDF outputs or Bitcoin block hashes. For instance, if two transactions are truly tied by all sorting criteria, the protocol can use the last VDF result (a pseudo-random value unknown until computed) to decide ordering in a fair way. This randomness is **unbiased** and **unpredictable** by participants (since no one can fast-forward the VDF or control Bitcoin block hashes), ensuring no incentive to grind or game tie situations. The outcome of such tie-breaks is consistent for all nodes (everyone computing the same VDF or seeing the same block hash gets the same result).

Pre-PoCtm Phase – On-Chain UTXO-Referenced Transactions

In the initial deployment phase ("Pre-PoCtm" – before the Proof-of-Continuum model is activated), every smart contract invocation is included directly in a Bitcoin transaction. In other words, **each OP_NET transaction rides inside a Bitcoin L1 transaction** in the same block. This ensures full compatibility with Bitcoin's existing mechanics and simple validation logic at the cost of slowness. Key characteristics of this phase:

- **Transaction Format:** An OP_NET contract call is embedded in a Bitcoin TX typically via an output script or witness data. For example, a user wanting to swap BTC for an OP_20 token would create a Bitcoin transaction that (a) has an input funding the gas fee and the BTC to swap, and (b) an output (or witness field) containing the OP_NET contract call parameters (such as "swap X amount of token for Y sats"). OP_NET leverages Taproot's flexible scripting and reveals the full data when spending that output in the same block or the next. By leveraging segwit's allowance for larger witness data, OP_NET can attach a contract call without bloating the UTXO set (the data is revealed in witness and thus does not remain as a UTXO).

- **UTXO-Referenced Execution:** Each OP_NET transaction may reference specific Bitcoin UTXOs that are relevant to the contract's execution. For instance, a contract that uses a user's BTC funds will list the outpoint of a UTXO that the user provides; that UTXO will be an input to the Bitcoin transaction, so it gets consumed on-chain, and simultaneously the OP_NET logic will know that amount of BTC is provided. OP_NET's **execution engine uses the presence of that UTXO and its unlocking signature as proof of the BTC being committed** to the contract. The OP_NET VM can then, for example, credit an equivalent amount of OP_20 tokens to the user or to another party, completing a swap. Because the Bitcoin transaction is the carrier, **the on-chain confirmation of that transaction is the moment the contract takes effect**. All OP_NET nodes, upon seeing a new Bitcoin block, scan each transaction for OP_NET markers. If found, they parse the contract call, validate its signature and gas, then execute the contract code in OP_VM. After execution, they update the OP_NET state (contract storage) accordingly. This tight coupling means if the Bitcoin transaction fails to confirm (e.g., due to a chain reorg or being outbid in the mempool), the contract call is considered void – nothing changes in OP_NET state because the trigger (the L1 TX) never completes.

- **Gas and Fee Handling:** Gas in OP_NET Pre-PoCtm is measured in computational units (gas) similar to Ethereum, but the **fee for gas is paid in BTC** via a mineable UTXO puzzle. Each contract call transaction includes an input (funded by the user) that pays for the execution. Gas pricing is based on an internal schedule calculated every block, and the mapping from gas units to satoshis is preset by the protocol (and could be adjusted via upgrades if needed). During execution, if a transaction would exceed the gas limit provided, it is halted and marked as failed. OP_NET nodes have to ensure the user isn't

trying to "cheat" by attaching too low a fee for a high gas operation – such a transaction might linger in Bitcoin mempools.

- **Execution Semantics:** In this phase, **every state change is directly linked to a specific Bitcoin TX**. That means OP_NET's global state is advanced in discrete steps at Bitcoin block boundaries. If Block 800000 contains 5 OP_NET transactions, the OP_NET node will process them one by one and update state. This yields the *same final state* on every honest node because Bitcoin's total order is definitive. Importantly, **atomicity is naturally ensured**: if a Bitcoin transaction moves some satoshis and also triggers an OP_NET token transfer, those two actions succeed or fail together. Either the Bitcoin tx made it on-chain (in which case the OP_NET token transfer was executed by all nodes) or it didn't (in which case no OP_NET node will apply that transfer). For example, suppose Alice wants to swap 1 BTC for Bob's 20,000 OP_20 tokens via an OP_NET DEX contract. In Pre-PoCtm, Alice would create a Bitcoin tx paying 1 BTC to Bob (or a DEX UTXO) and carrying the contract call; when that tx is confirmed, Bob's address now has 1 BTC on L1 and Alice's OP_NET account is credited 20k tokens by the contract logic. If that tx is later orphaned in a reorg, Bob's BTC return to Alice (since the tx disappeared), and OP_NET nodes **automatically revert** the token transfer (back to Bob) as they roll back to the previous block's state. They do this by keeping track of historical state changes tied to each Bitcoin block (a mini "state checkpoint" per block). On reorg, OP_NET nodes undo the changes from orphaned blocks and reapply those from the new fork of blocks, keeping OP_NET state in sync with Bitcoin at all times. This tight coupling to Bitcoin's ledger guarantees that no contract state can drift out of agreement with on-chain reality.

- **Limitations:** The Pre-PoCtm mode, while simple, has scaling limitations. Every contract invocation takes up space in a Bitcoin block. Throughput and latency are limited by Bitcoin's ~10-minute blocks and ~4MB weight limit. Nonetheless, this phase establishes the foundation and security model: OP_NET behaves like an L1 extension of Bitcoin, with full Bitcoin security. It sets the stage for the improvements in the next phase by demonstrating **end-to-end atomic swaps (BTC <-> tokens)**, on-chain verification of contract calls, and the basic node software for executing and verifying contracts. During Pre-PoCtm, developers can deploy tokens (using the OP_20 standard, akin to ERC-20 on Bitcoin) and dApps, with the assurance that each action is mined on L1 (making it as secure as a Bitcoin transaction, albeit at Bitcoin's throughput). This phase essentially functions as a "beta" with **maximal security** but lower performance, allowing the community to battle-test OP_NET's contract VM and UTXO handling in a straightforward environment.

Post-PoCtm Phase – Ephemeral Sub-Blocks & Proof-of-Continuum

Once OP_NET has proven stable, the protocol will transition to the Post-PoCtm phase, activating the **Proof of Continuum (PoCtm) consensus model**. In this phase, OP_NET decouples most contract execution from the immediate inclusion in L1 blocks. Instead, *ephemeral sub-blocks*, or **slots**, are produced at a higher frequency (e.g. several per Bitcoin block interval), and only summary data is anchored on-chain. This dramatically increases throughput and reduces latency for contract confirmations, while still relying on Bitcoin for final settlement and security. The components of this phase include:

Ephemeral Slots and Internal State Updates: Time is divided into slots (sequence numbers) governed by the VDF. At the start of each slot, nodes collect all pending OP_NET transactions from the network (the OP_NET mempool). Using a deterministic algorithm, they **order these transactions** by: (1) ascending gas usage (small computations first), (2) descending priority fee (higher-paying tx get precedence if gas usage is equal), and (3) if still tied, by a VDF-based random tie-breaker. This ensures a fair ordering that **cannot be manipulated by any single node or miner** – even if someone has a high fee transaction, they cannot preempt smaller transactions that have already been queued, which mitigates MEV (Miner Extractable Value) opportunities. Gas-first ordering prevents a single large-gas tx from delaying many small tx; it favors including as many tx as possible and improves parallelism. Fee-second ordering still rewards users who pay more within the same gas class. And the VDF tie-break means if two users somehow craft identical size and fee, the outcome is essentially random and not knowable in advance, so there's no strategy to always win ties.

Once ordered, **each node independently executes the transactions in that slot in the OP_VM**, applying all state changes in sequence. Because the ordering is deterministic and execution is deterministic, all honest nodes that have the same set of pending transactions will end up with the same new state for the slot. Notably, this does not require a *leader* – it's a form of *simultaneous consensus* where the rules themselves decide the block content. In practice, there may be slight differences if some node is missing a transaction that others had; however, the consensus layer will resolve that at anchor time (we discuss this under Anchoring and Fork Choice below). For now, assume a well-connected network where all valid tx propagate to all nodes by the end of the slot (with the slot duration tuned to network latency).

Each slot's execution produces a **State Delta**: essentially, the collection of all changes (accounts debited/credited, contract storage modified, events emitted) and the new state root hash. It also produces a **Transaction Merkle Tree**: all transactions processed in the slot are hashed and aggregated into a Merkle root. Together, this info (state root + tx root) constitutes a *block commitment*. The node that finishes execution can produce a PoC proof and broadcast it to the network.

VDF-Based Timing (Proof of Continuum): The length of each slot is enforced by a Verifiable Delay Function. For example, suppose we want slots of 10 seconds. At the end of slot N, a node takes some agreed-upon seed (e.g. the hash of the last Bitcoin block or last slot's data) and begins computing the VDF for 10 seconds of work. Only after completing this computation (and

getting a VDF proof) can the node move on to finalize slot N+1. In effect, **the VDF simulates a predictable clock** for the protocol – every slot will take at least X seconds, no matter how much computing power one has. This prevents a malicious actor from attempting to generate many sub-blocks rapidly to outrun others. All nodes are doing the same sequential work; a faster CPU might finish a bit earlier, but not by orders of magnitude (assuming similar hardware). The VDF output for slot N can also be used as a source of randomness (for tie-breaks as mentioned, or other in-protocol needs like contract random number generation if exposed).

The concept that "time passes via a verifiable function" is the essence of the term ***Proof of Continuum***: it ensures the continuum of time and order in the sequence cannot be distorted. Each slot's VDF proof is posted along with the anchor, so anyone can verify that the prescribed delay between slots was respected (the proof attests that a certain number of sequential steps were done, hence at least that much wall-clock time passed).

Ephemeral State and Execution: During slot execution, transactions can do everything they did in Pre-PoCtm (transfer tokens, call contracts, etc.), except directly finalize Bitcoin transfers. If a transaction in a slot wants to interact with a Bitcoin UTXO (e.g. spend a BTC UTXO as part of a swap), it can – **the user must have provided the UTXO reference and signature in the transaction** – but the actual consumption of that UTXO will only occur when the anchor transaction is mined. In the interim, OP_NET nodes mark that UTXO as *reserved/spent in OP_NET*. They will refuse any other OP_NET tx that tries to use the same UTXO (preventing double-spend in the OP_NET layer). If by chance an unrelated Bitcoin transaction spends that UTXO before the anchor, then the anchor when constructed will fail to include it (since it's gone), and OP_NET will eventually revert the effects of the transaction that depended on it (as it can't be fulfilled). While this is a corner case, it underscores that **the final authority on UTXO existence is the Bitcoin chain** – OP_NET must always defer to it. In most cases, users will not double spend their own input; and no one else can, because they don't have the key.

Anchoring Ephemeral Blocks: The link between the execution and on-chain finality is the anchor transaction. The simplest strategy is **one anchor per Bitcoin block**, committing all slots since the last block. For example, if 3 slots occurred since the last Bitcoin block, the next anchor will include individual roots of all transactions from those 3 slots and the final state after those slots.

When a Bitcoin block is mined, all OP_NET nodes perform **anchor validation**: they take the Merkle root from the anchor (OP_NET transactions in that block), compare it to their own calculated root for those slots. If it matches, great – the consensus and on-chain consensus agree. The OP_NET state is now finalized up to that slot. If it *doesn't* match, that means the anchor is committing to a different set of transactions or outcome than this node expected. Nodes will then retrieve the transactions and proofs corresponding to that root and verify them. **There are a few scenarios**: **(a)** a node might have missed some transactions that were included – in that case, the anchor is actually valid (the transactions were legit) and the node will update its state to include them (replaying the slot with those tx). Because the anchor provides an authoritative record (through inclusion proofs) , honest nodes converge on the anchored state. **Or (b)** the anchor could be malicious/incorrect – e.g., committing a transaction that doesn't

have a valid signature or a completely fabricated state. In that case, no honest node's execution would match it, and the discrepancy is evident. **However, Bitcoin has still accepted the transaction** (since Bitcoin miners only care that the anchor TX itself is valid as a Bitcoin TX, not what the OP_NET data means). **This situation is prevented by the PoC proofs** – e.g. if OP_NET recomputes the transactions, an invalid state root wouldn't have a valid proof and honest nodes would reject the anchor as "illegitimate" even if it's in a Bitcoin block. Nodes would treat it as an attack and wait for a subsequent anchor (perhaps produced by honest parties) that overrides it. **This is a complex scenario – essentially a fork of the OP_NET state despite the Bitcoin inclusion.** In practice, it's expected to be exceedingly rare because a malicious anchor can't actually steal funds (they can't fake signatures for BTC outputs, so they can't grab users' coins; at best they could exclude some tx, or include an invalid tx that doesn't affect on-chain outputs).

If they exclude transactions, that *is* a breach of protocol – OP_NET nodes would spot that immediately (the Merkle tree wouldn't include the legitimate tx). The deterrent is that Bitcoin miners have no incentive to include a fraudulent anchor that the network will reject – they only get the fees if they include the users' inputs anyway, and doing so correctly vs incorrectly yields the same fees. So we assume anchors will be honest or the network will swiftly insist on correction.

Fork Choice and Finality: OP_NET's fork choice (which state is canonical) is 100% tied to Bitcoin's longest chain. **There is no separate fork resolution needed** – if Bitcoin reorgs, OP_NET reorgs the same blocks. If two competing anchors appear in different Bitcoin forks, whichever Bitcoin fork wins will determine which anchor is accepted. Typically, miners will include only one because the anchor uses a specific UTXO, so a second would be a double-spend of the same mineable UTXO and thus not included. So, the rule is: **follow the Bitcoin chain, and for each block take at most one anchor commitment (the anchor with the latest data).** If a block has none (maybe no activity or a miner ignored it), the OP_NET state simply doesn't finalize new changes that period. In such cases, pending slots can roll over to the next block's anchor. OP_NET nodes might even attempt CPFP (Child-Pays-For-Parent) via the mineable UTXO to get an anchor confirmed if a miner forgot it – but because the anchor UTXO is anyone-can-spend, any node can fee-bump it by attaching a high-fee child, effectively ensuring it will get confirmed. Thus, censorship of anchors is hard – anyone can republish the anchor if miners ignore it, by increasing fees.

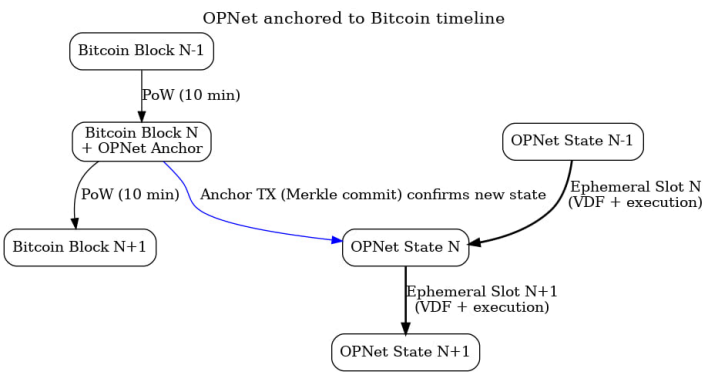

Figure: Timeline of OP_NET ephemeral slots and anchoring. state (right side) progresses through slots (controlled by VDF delays) from State N-1 to State N to N+1. Each Bitcoin block (left side) contains an anchor transaction committing to the new state (blue arrow). This anchor finalizes all the

transactions for that interval, linking the OP_NET state progress to Bitcoin's PoW timeline.

In summary, during Post-PoCtm the **bulk of contract execution is handled** by the network of nodes, achieving higher speed, and then periodically **snapshotted on-chain via a single compact transaction**. This design is akin to a rollup: it inherits Bitcoin's security for final settlement, but does computation. Unlike traditional sidechains or rollups, OP_NET's approach does not rely on a centralized sequencer or a multi-day challenge period – it uses Bitcoin miners (via anchors) and cryptographic delays to **trustlessly coordinate transaction ordering**. By leveraging Bitcoin L1 directly for ordering and validation, OP_NET avoids new consensus vulnerabilities: it effectively uses Bitcoin's miners as the "block proposers" (they include the anchor) and **anyone as the block builders** (the nodes propose the content). The result is a decentralized, permissionless protocol where **Bitcoin provides the spine of truth** and OP_NET provides a flexible, high-level execution layer.

State Management and OP_VM Execution

OP_NET maintains its own internal state database to track user accounts, token balances, contract code and storage, and other metadata. This state is separate from Bitcoin's UTXO set but is always derived from and anchored to it. Key points of state management:

- **Account Model with Unified Keys:** OP_NET uses a unified account model where each participant is identified by a **Tweaked Public Key** (derived from their Bitcoin key). This means a single keypair can control BTC UTXOs on L1 and also serve as the address for OP_NET tokens and contracts. Accounts hold balances of native BTC (which might represent BTC that have been pegged into OP_NET contracts), OP_20 tokens, and possibly other assets or NFTs (using OP_721 standard). Contracts themselves also have addresses (often created/determined by a special transaction, analogous to contract creation in Ethereum). The use of tweaked keys and Schnorr signatures allows OP_NET contracts to verify signatures **natively** – a contract can include logic like `require(sigValid(pubkey, message))` to enforce permissions, leveraging the same crypto as Bitcoin.

- **Contract State and Storage:** Each smart contract may hold key-value storage (for example, a mapping of addresses to balances in a token contract, or more complex structures for DeFi apps). OP_NET organizes contract storage in a Merkleized structure so that a global state root can be computed. However, OP_NET does not currently publish the entire state root to Bitcoin. Instead, it relies on the fact that all transactions are recorded and can be replayed to derive state. The state root is used within the protocol for quick syncing and cross-checks: nodes can exchange asserted state roots for a given block height to ensure consistency (if they diverge, some transaction processing differs, indicating an issue).

- **State Persistence and Snapshots:** OP_NET nodes persist the full state (accounts, balances, contract storage) in local databases. After each ephemeral slot or each Bitcoin block (depending on phase), the node may take a snapshot of the state (or at least the state root). This aids in quick rollback during reorgs – a node can revert to the snapshot

of the previous confirmed block and re-process, rather than re-running everything from genesis. For safety, nodes typically keep a rollback journal for the last few Bitcoin blocks (since deep reorgs are extremely rare beyond e.g. 2-3 blocks). During normal operation, the latest state is considered **confirmed** once its anchor is buried 6 blocks deep on Bitcoin (just as one would treat 6-conf as final).

- **OP_VM (WASM Execution Engine):** The OP_NET Virtual Machine executes contract code compiled to WebAssembly (WASM). Developers can write contracts in high-level languages (Rust, C, AssemblyScript), compile to WASM, and deploy on OP_NET. The VM is deterministic (ensuring every node gets the same results given the same input), and it meters execution to charge gas. It has access to a set of **syscalls/opcodes** for blockchain-specific functions: e.g., reading the current block height, emitting logs/events, etc. Importantly, the VM can also verify Bitcoin signatures as noted, and handle hashing, which is essential for things like checking Merkle proofs or implementing custom crypto within contracts. Gas costs are assigned per opcode in a way to roughly reflect computational cost (e.g., a SHA256 hash or a big elliptic-curve verify will cost a higher gas amount). The OP_VM is designed to be **upgradable**: new instructions or precompiles can be introduced via protocol upgrades to support features like pairing-based crypto, SHA-3, or others in the future. The use of WASM means the system isn't tied to a single language or a custom bytecode – it leverages a well-understood, sandboxed execution environment that is already optimized in many clients.

- **Gas and Resource Management:** In the ephemeral context, OP_NET must be careful with resource usage since transactions aren't immediately constrained by Bitcoin block size. To prevent spam or DoS in slots, each slot will still have a **gas limit** (analogous to block gas limit in Ethereum). This ensures one slot's worth of execution is bounded (so even if it's , it doesn't overload node CPUs or memory). The exact numbers will be tuned through testing. Additionally, transactions may have dependencies on state – OP_NET handles these similarly to Ethereum: if Tx B reads something modified by Tx A in the same slot and comes later in order, it will see A's effects. If Tx A fails (reverts), it still consumes its gas but its state changes are undone, and subsequent to proceed as if A had no effect (except gas usage). The Merkle tree commitment will include failed tx as well (to prove they were included even if they didn't change state).

- **Parallelism and Future Optimizations:** The deterministic ordering presently implies single-threaded execution of transactions per slot (ensuring sequential consistency). In the future, OP_NET will allow parallel execution of independent transactions (those touching different contracts/state) to speed up processing, as long as the end result is identical to some sequential order (this is an area of research and would be an internal optimization not affecting consensus results). The protocol's design doesn't preclude this – it just requires that the Merkle root reflects all transactions as if in some order, which parallel execution would still produce.

- **State Access to Bitcoin Data:** Within OP_NET contracts, there may be a need to reference Bitcoin layer data. For example, a contract might want to know the current **Bitcoin block height or hash** (for randomness or external triggers). OP_NET provides a way to supply such info: the anchor transaction's inclusion in a Bitcoin block inherently ties the OP_NET state to a specific Bitcoin height. A contract can query "current Bitcoin block height" and the node will return the height of the last processed anchor. For randomness, one could use the Bitcoin block hash (which is unpredictable to contract users in advance) – OP_NET could make blockHash(n) available for recent n.

  Caution: direct use of block hash can be risky due to miner manipulation on a small scale. Alternatively, VDF outputs provide an intrinsic randomness as well.

- **Fees and Incentives in Post-PoCtm:** A subtle aspect is how miners are incentivized to include anchors if some of the fee outputs are in the anchor TX and mineable UTXOs. In practice, as mentioned, the anchor TX will carry the aggregate fee. Miners will include it if that fee is competitive. Since it includes all users' fees, it should be quite large relative to normal transactions (imagine 100 OP_NET tx paying 1 sat/vByte equivalent each – together they can offer a sizeable fee to one anchor TX). There is a risk if Bitcoin blocks are empty or low fee, an anchor could be delayed; but OP_NET transactions often involve value transfers and users can always attach higher fees if they need fast confirmation.

  Additionally, OP_NET might implement a secondary incentive: *Anchor bounties.* For instance, the protocol could decide that a small fraction of gas fees are "tips" specifically for the address that constructs the anchor. This would reward the node that assembles and publishes the anchor (covering their bandwidth/effort) on top of the miner's reward. However, since any node can step in to do it, this is optional. More importantly, this phase introduces the idea of **fee rebate or slashing for incorrect anchors** – if someone posts a bad anchor, they might lose some bond. But given no staking, the main penalty is just that a malicious anchor's included fees might be burned or not retrievable by the attacker (they paid fees for bogus data).

  Overall, the incentive structure aligns: users pay fees to get their tx included, miners collect fees by including anchors, and honest nodes ensure the anchors are correct to keep the system usable (thus preserving the flow of fees in future).

Validator Node Behavior and Security Guarantees

Node Roles: All **full** nodes in OP_NET perform the same functions: **they track the Bitcoin blockchain**, maintain the OP_NET state, and execute new transactions. There is no special role like "miner" or "validator" separate from being a full node – every node can produce blocks (propose anchors) and every node validates blocks. Some nodes may focus on certain tasks (for example, a well-resourced node might specialize in running the VDF as fast as possible to be first to propose anchors, effectively acting as a block builder, while others just verify). But the protocol does not assign unique leader roles; **it's an *optimistic concurrency*** model where nodes process transactions and the first to get the anchor wins, but everyone must verify it.

Networking: OP_NET nodes communicate over a P2P network (similar to Bitcoin's). Transactions are gossiped to all nodes. To avoid abuse, transactions might be relayed only if they meet some minimum fee/gas criteria (prevent flooding of zero-fee tx). Ephemeral block proposals (like a list of transactions and resulting state root) might also be gossiped among nodes before anchoring, to converge on the result. However, since the anchor ultimately decides, nodes could just locally compute and wait for the anchor rather than share intermediate results. Sharing could speed up detection of malicious activity (if one node sees another propose a different root, it knows a disagreement exists). The network likely includes some **bootstrap peers** (hardcoded or known via DNS seeds [**not yet implemented**]) to help new nodes quickly get the OP_NET data (since Bitcoin blocks only have commitments, a new node syncing from genesis would need the full history of OP_NET transactions to rebuild state). These bootstrap nodes can provide snapshots or the full log of OP_NET tx with cryptographic assurances. The snapshots would include the state root and could be verified against some trust (or recomputed from the log). While a new node *could* sync by reading every Bitcoin block and extracting OP_NET data (fully trustless but slow), the bootstrap nodes' presence expedites adoption. **Importantly**, trusting a snapshot is not mandatory – a cautious user can verify it by cross-checking random inclusion proofs against the on-chain commitments or fully recomputing critical sections. The design is such that **any user can be fully self-verifying** given enough time and resources, just like in Bitcoin (you can start from the genesis UTXO and verify every transaction).

Validation Process: Whenever a Bitcoin block is found, the OP_NET node does:

1. **Scan the block for OP_NET anchor TX:** If found, verify that it uses the previous anchor's in its data and extract the Merkle root and any proofs. Use inclusion proofs to obtain any transactions that might not have been in the mempool (linked by UTXOs, spent.) (this requires either the anchor TX carried them or they are fetched from the proposing node). Validate the PoC proof (if any) attached. If any of these checks fail (e.g., anchor doesn't spend the correct UTXO, or the proof is invalid), mark the anchor as invalid. (The node would likely then wait for a proper anchor in a future block, or if an alternate valid anchor also appears in the same Bitcoin block, which is **unlikely** due to double-spend rules, prefer the valid one).
2. **Apply Confirmed Transactions:** For each OP_NET transaction that was anchored (either individually in Pre-PoCtm or via the batch in Post-PoCtm), apply its effects to the state if not already applied. In Pre-PoCtm, "applying" means executing the contract now (though some nodes might pre-execute when they saw it in mempool, they will re-run to be certain and use the block ordering). **In Post-PoCtm**, if the node had already processed slots internally, **it just needs to finalize that state**. If the anchor included something unexpected (like a tx that the node hadn't processed), the node will fetch and execute it now, bringing its state in line with the anchored result. Essentially, the Bitcoin block serves as a checkpoint: the node reconciles its speculative state with the official state, then commits that as final.
3. **Revert any orphaned state:** If this block caused a reorg (i.e., some previous Bitcoin blocks were orphaned), the node will roll back the OP_NET state to the last common

block with the new chain. This involves undoing the state changes of those orphaned blocks' OP_NET transactions. Thanks to snapshots, the node can load the state at the fork point. Then it applies the new blocks' anchored transactions as above. This ensures that even deep reorgs (though rare) are handled consistently: OP_NET state always reflects the active Bitcoin chain's sequence of anchors. All transactions that were in orphaned blocks return to the mempool as unconfirmed (they could be re-included by the next anchor if still valid).

4. **Advance to next slot (Post-PoCtm):** After incorporating the latest block, the node knows the new anchor UTXO and possibly gets a new seed (block hash) for randomness. It will then continue processing new ephemeral transactions for the current open slot until the next anchor is due.

Security Analysis: OP_NET's security comes from a combination of Bitcoin's security and cryptographic guarantees within the protocol:

- ***Double-Spend and Reorg Protection**:* By tying OP_NET state transitions to Bitcoin confirmations, it inherits Bitcoin's resistance to double-spending. No OP_NET token transfer or contract execution is considered final until the underlying Bitcoin triggers (anchors or UTXO spends) are final on **L1**. If a Bitcoin reorg happens, OP_NET might transiently have shown a transaction as executed, but it will auto-revert it – this is the same eventual consistency Bitcoin itself has. **Users should treat OP_NET transactions as final after the same number of confirmations they trust for Bitcoin transactions** (e.g., 6 blocks). Within a single slot (pre-anchor), a user might see a pending result, but they must understand **it's not immutable** until **anchored**. This is analogous to seeing an Ethereum transaction in a pending block before it's mined – **likely but not guaranteed.**

- ***No Validator Trust**:* The design pointedly avoids requiring trust in any "committee" or master nodes. Even though one node might propose a block, everything that node does is verified by all others through the anchor commitments. There's no need to trust that node's honesty or stake – if it cheats, the worst it can do is get an invalid anchor in a block, which others will detect and ignore in terms of state update (though the Bitcoin chain will still contain the TX). There are no slashing or slashing conditions because we don't lock up any stake; **the security relies on economic incentives** (**miners wanting fees, users not wanting to waste fees, honest nodes outnumbering or outlasting attackers in broadcasting correct anchors**). It's assumed that a majority of Bitcoin miners by hashpower are indifferent-but-honest about including valid anchors (they'll include whatever pays fees and follows Bitcoin rules). **Even if a majority tried to censor OP_NET**, **OP_NET would employ countermeasures like channeling anchors through transactions that look normal or via collaborative fee-bumping to slip anchors in**. The system has **no central point that selects or orders transactions** – order comes from either predetermined rules or the outcome of a fair race (VDF + miner inclusion). This decentralization maximizes censorship-resistance.

- ***Cryptographic Verifiability**:* Every aspect of state transition is verifiable by a third party with access to the blockchain and (if needed) the P2P data. For any given final state, a

light client could ask a full node for a Merkle proof of a particular account's balance against the known state root (if state root is known). Or, the client can verify inclusion of a particular transaction in an anchor via Merkle proof, **and then verify the execution outcome of that transaction by reading the contract code (if the code is open) and re-running it on the provided inputs (this is heavy for a light client, but possible).**

In the future, OP_NET will integrate validity proofs for each anchor (an upgrade), **a light client will simply verify the succinct proof and know the state is correct without re-executing**. This will provide near-total certainty of correctness even if most of the network were malicious. Even without that, the worst-case scenario for a user who doesn't run a full node is to trust a couple of independent full nodes to provide the same state info – since no token is at stake, these full nodes have no reason to collude and lie (and any lie would be caught by Bitcoin data anyway). **This model is similar to how one trusts multiple blockchain explorers or community run nodes to cross-verify data.**

- ***Handling of Invalid States**:* Suppose a bug or attack leads to an invalid state being committed (*somehow an anchor with a bad state root got buried and accepted by some nodes*). **In such an extreme case**, the community would likely coordinate out-of-band to patch the node software (fix the bug or blacklist that anchor) and follow the correct state (effectively a user-activated soft fork of OP_NET rules). T**his is comparable to how Bitcoin would handle a consensus bug** – not ideal, but part of being future-proof is having governance processes to remedy unforeseen issues. Since OP_NET has no token, governance is informal (developers and node operators agreeing on fixes). *Upgrades to the protocol (like introducing proofs or changing gas costs) would similarly be enacted by releasing new node software and reaching rough consensus to adopt it at a specific block height.*

In conclusion, when the Post-PoCtm phase is active, OP_NET achieves a delicate balance: **Bitcoin anchors enforce global consistency and finality, while slots provide scalability and speed**. The system is as secure as Bitcoin for final outcome, and interim operations are secured by cryptographic constraints that greatly limit an attacker's freedom of action. There is no **new** crypto-economic assumption like a stake majority – just the assumption that Bitcoin's proof-of-work remains robust and that cryptographic hash and delay functions are not broken.

<u>If those hold, OP_NET can truly function not only as a decentralized L1 smart contract platform for Bitcoin but also increases Bitcoin scalability and speed.</u>

Conclusion

This specification outlined how OP_NET operates from the ground up: starting with direct Bitcoin transactions and evolving into a sophisticated execution layer anchored by Bitcoin and providing a viable solution to true decentralized speed. By combining **deterministic execution rules**, **Merkle commitments**, and a **VDF-driven continuum of time**, OP_NET achieves a decentralized consensus without new trusted parties. All critical data is ultimately stored or committed on Bitcoin L1, meaning the integrity of OP_NET is always verifiable against the Bitcoin ledger by anyone. The design prevents double-spending and replay attacks by synchronizing with Bitcoin's state and ensures atomic operations, fulfilling the vision of Bitcoin-backed smart contracts.

Crucially, OP_NET does this in a **Bitcoin-aligned way** – no modifications to Bitcoin are required, and miners and nodes can incrementally opt into OP_NET for additional utility and revenue. **It complements Bitcoin**: Bitcoin provides secure ordering and **finality**, while OP_NET provides flexibility and expressivity. The protocol is **trust-minimized** (relying only on standard cryptographic assumptions and Bitcoin's consensus) and avoids any form of centralization (no federations, no admin keys, no privileged oracles, no multisigs). All participants – whether they run a node or not – can have high confidence in the system because of its cryptographic auditability and the game-theoretic alignment with Bitcoin's incentives.

As OP_NET matures, it could effectively transform Bitcoin into a full-fledged smart contract platform while **retaining Bitcoin's unmatched security and decentralization**. ***By anchoring a new continuum of computation onto the oldest and most secure blockchain, OP_NET opens the door for decentralized finance, asset issuance, and complex applications to thrive on Bitcoin without sacrificing the core principles that make Bitcoin trusted.*** Each layer of OP_NET is carefully engineered to be robust yet adaptable, ensuring that the protocol can stand the test of time much like Bitcoin itself.

Through OP_NET, Bitcoin L1 becomes not just the settlement layer for sound money, but also the **foundation for a world of trustless applications** – all secured by Satoshi's original network. This specification serves as a technical roadmap for that vision, describing how, with current technology, we can achieve a provably secure, decentralized consensus and execution framework on Bitcoin, and how we can iterate on it for future improvements. The result is a system that is **by Bitcoin, for Bitcoin, and with Bitcoin** at its heart, fueling every computation with BTC and recording every outcome on the Bitcoin ledger, thereby inheriting the strength and trust of the Bitcoin ecosystem.