The Universal Protocol & Simplicity Indexer: A Foundation for Scalable Token Infrastructure on Bitcoin

lamachina (monsieur.foo)

The best protocol is the one we build together—block by block.

June, 2025

Abstract

The Universal Protocol represents a fundamental evolution of the BRC-20 token standard on Bitcoin, addressing critical limitations in transfer logic, on-chain efficiency, and protocol extensibility while maintaining full backward compatibility. By migrating operational logic from witness inscriptions to OP_RETURN outputs—Bitcoin's native, prunable data carrier—the protocol achieves a "Satoshi-faithful" design that prioritizes long-term network health and decentralization. This architectural shift enables atomic multi-operation transactions, reduces blockchain bloat, and establishes the most robust accountability system for Bitcoin-native tokens. Through the Simplicity Indexer and a community-driven governance framework (Operation Proposal Improvements), the Universal Protocol provides a secure, scalable foundation for fungible token operations on Bitcoin's base layer, honoring the simplicity that drove BRC-20's adoption while resolving its foundational flaws.

Keywords: Bitcoin, BRC-20, OP_RETURN, Token Protocol, Simplicity Indexer, Universal Protocol, Fungible Tokens, OPI, Account-Based Model

Contents

1 Introduction

In early 2023, the Bitcoin ecosystem witnessed an explosion of innovation, sparked by the Ordinals theory. From this movement, the BRC-20 standard emerged, unleashing a wave of creativity and participation unlike anything seen before. Its success was rooted in a radical simplicity: a straightforward set of JSON operations—deploy, mint, transfer—inscribed into witness data, made it accessible to all. For the first time, a fungible token protocol on Bitcoin achieved massive, grassroots adoption.

Yet, the very simplicity that fueled BRC-20's meteoric rise also revealed its inherent limitations. As the ecosystem matured, these growing pains became apparent: transfer logic was ambiguous, creating vulnerabilities; on-chain data was stored inefficiently; and the protocol lacked a clear path for extension without introducing breaking changes. For the past years, the community has grappled with these challenges, understanding that the question was not *if* BRC-20 should evolve, but *how*.

The Universal Protocol emerges as the answer to that question. It is not a replacement, but a profound extension—one that honors the original BRC-20 syntax while resolving its foundational flaws. The core architectural decision is a migration of operational logic from witness inscriptions to OP_RETURN outputs. This choice is deliberate and profound. As Bitcoin's native, on-chain data carrier, OP_RETURN provides a surface-level, easily interpretable canvas for protocol logic. Crucially, unlike witness data, OP_RETURN outputs are prunable, meaning they do not contribute to the permanent UTXO set bloat that full nodes must maintain forever. This "Satoshi-faithful" approach prioritizes the long-term health and decentralization of the Bitcoin network.

By building upon this foundation, the Universal Protocol establishes the most robust accountability system in existence for a token protocol on Bitcoin. It maintains full backward compatibility with the original BRC-20 shared namespace and offers a tribute to Satoshi through its OPI-000 `no_return` mechanism. The intent, the action, and the outcome are explicitly and immutably linked on-chain. This document specifies that evolution.

Welcome to the Universal Protocol.

1.1 Contributions

This work makes the following key contributions to Bitcoin token infrastructure:

1. **OP_RETURN-Based Architecture:** A formal specification for migrating BRC-20 logic to prunable OP_RETURN outputs, reducing permanent blockchain bloat while maintaining full expressiveness.

2. **Account-Based Safety Model:** A robust token accounting system that ties balances to addresses rather than UTXOs, eliminating accidental token loss scenarios.

3. **Atomic Multi-Operation Transactions:** Native support for batching multiple token operations in single Bitcoin transactions, enabling trust-minimized swaps and significant fee reduction.

4. **Governance Framework (OPI):** A structured, community-driven process for protocol evolution inspired by Bitcoin Improvement Proposals.

5. **Reference Implementation:** The Simplicity Indexer, a deterministic state machine for validating and tracking Universal Protocol operations.

2 Fundamental Concepts

The Universal Protocol achieves BRC-20 functionality through a lean and explicit transaction structure, interpreted by the Simplicity Indexer.

2.1 Transaction Architecture

A Universal transaction is composed of four key elements:

1. **Inputs (UTXOs):** Standard Bitcoin UTXOs that provide the necessary BTC for network fees. For transfer operations, the Simplicity Indexer associates a specific address from input UTXOs with token balances, thereby authorizing the transaction.
2. **OP_RETURN Output(s):** The core of the protocol. These are non-spendable outputs containing a BRC-20 operation as a compact JSON payload ($\leq$ 80 bytes). A single Bitcoin transaction can contain multiple OP_RETURN outputs to enable batch processing.
3. **Recipient Output(s):** Standard Bitcoin outputs (P2PKH, P2SH, Taproot). The protocol's central design principle is that each BRC-20 OP_RETURN operation is explicitly bound to the first valid, non-OP_RETURN output that immediately follows it. The address of this subsequent output is the sole recipient of the tokens for that operation.
4. **Change Output:** A standard output returning any remaining BTC to the sender's address.

2.2 The Account-Based Model: A Foundation of Safety

The Universal Protocol employs a robust account-based model, a design choice that fundamentally distinguishes it from many UTXO-based token systems. In this model:

- A user's token balance is tied directly to their Bitcoin address, not to a specific, fragile UTXO.
- This eliminates the risk of accidental token loss. Spending a UTXO from an address that holds tokens for a standard BTC transaction will not affect the token balance. The tokens remain securely associated with the address.
- An explicit Universal transfer operation is the only way to debit tokens from an address's balance. This provides a safe, intuitive, and forgiving user experience.

This architectural choice represents a fundamental safety improvement over UTXO-bound token systems, where careless UTXO management can lead to irreversible token loss.

2.3 Design Principles

The Universal Protocol is guided by four core principles:

1. **Explicit Binding:** Every operation is explicitly bound to its recipient through deterministic output ordering, eliminating ambiguity.
2. **Prunable Efficiency:** Use of OP_RETURN ensures protocol data does not contribute to permanent UTXO set growth.
3. **Backward Compatibility:** Full interoperability with existing BRC-20 deployments through shared namespace recognition.
4. **Structured Evolution:** Community-driven governance (OPI) enables controlled protocol extension without breaking changes.

3 Operation Specifications

All operations are defined by a JSON payload within an OP_RETURN output. Each operation has a specific, mandatory transaction structure that the Simplicity Indexer validates.

3.1 Deploy Operation

The `deploy` operation registers a new BRC-20 token and its properties.

```
{
  "p": "brc-20",
  "op": "deploy",
  "tick": "OPQT",
  "max": "21000000",
  "lim": "1000"
}
```

Parameters:

- `p`: Protocol identifier; must be `"brc-20"`.
- `op`: Operation type; must be `"deploy"`.
- `tick`: The unique ticker symbol for the token.
- `max` (or `m`): A string representing the maximum total supply.
- `lim` (or `l`): An optional string representing the maximum amount per mint operation.

Validation: The Simplicity Indexer will only validate a deploy if the tick has not been previously registered by any valid BRC-20 deployment (Ordinals or Universal), enforcing a shared global namespace. The JSON payload must not exceed the 80-byte OP_RETURN data limit.

Transaction Structure:

Component	Description	Notes / Rules
Input(s)	Standard UTXO(s)	Provides the BTC required for transaction fees.
Output 0	OP_RETURN	Payload: `{"p":"brc-20","op":"deploy",...}` MUST be the first output (vout=0).
Output 1	"Dummy" Recipient	Address: Sender's own change address (recommended). Value: $\geq$ Dust Limit (e.g., 546 sats). This output is structurally required but semantically null.
Output 2	Change (Optional)	Address: Sender's change address. Value: Remaining BTC.

Table 1: Deploy Operation Transaction Structure

3.2 Mint Operation

The `mint` operation creates new units of an existing token. This operation credits the `amt` of the specified `tick` to the address of the Bitcoin output that immediately follows this OP_RETURN output.

```
{
  "p": "brc-20",
  "op": "mint",
  "tick": "OPQT",
  "amt": "1000"
}
```

Transaction Structure:

Component	Description	Notes / Rules
Input(s)	Standard UTXO(s)	Provides the BTC required for transaction fees.
Output 0	OP_RETURN	Payload: `{"p":"brc-20","op":"mint",...}` MUST be the first output (vout=0).
Output 1	Token Recipient	Address: The address receiving the newly minted tokens. Value: $\geq$ Dust Limit (e.g., 546 sats).
Output 2	Change (Optional)	Address: Sender's change address. Value: Remaining BTC.

Table 2: Mint Operation Transaction Structure

3.3 Transfer Operation

The `transfer` operation moves existing tokens from a sender to a recipient. This operation debits the `amt` from the sender's balance (as identified by the transaction inputs) and credits it to the address of the Bitcoin output that immediately follows this OP_RETURN.

```
{
  "p": "brc-20",
  "op": "transfer",
  "tick": "OPQT",
  "amt": "100"
}
```

Transaction Structure:

Component	Description	Notes / Rules
Input(s)	Sender's UTXO(s)	Crucial: The inputs serve to authorize the transaction and identify the sender's address. The indexer then verifies the token balance of the entire address, not a specific UTXO.
Output 0	OP_RETURN	Payload: `{"p":"brc-20","op":"transfer"}` MUST be the first output (vout=0).
Output 1	Token Recipient	Address: The address receiving the transferred tokens. Value: $\geq$ Dust Limit (e.g., 546 sats).
Output 2	Change (Optional)	Address: Sender's change address. Value: Remaining BTC.

Table 3: Transfer Operation Transaction Structure

4 Multi-Operation Transactions

A core feature of the Universal protocol is the ability to batch multiple, distinct operations within a single atomic Bitcoin transaction. This is achieved by sequencing OP_RETURN and recipient outputs, leading to significant reductions in network fees and enabling powerful, trust-minimized functionalities.

The Simplicity Indexer processes the outputs of a transaction sequentially. This guarantees that operations within a single transaction are resolved in a precise order, allowing the outcome of one operation (like a deploy) to be the prerequisite for the next (like a mint).

4.1 Standard Multi-Transfer

This is the most straightforward batching use case, where a single sender distributes multiple tokens to various recipients.

Scenario: A user wants to transfer 100 "OPQT" tokens to Alice and 10 "ORDI" tokens to Bob in one transaction.

Transaction Output Structure:

vout1: Input(s): From the sender, containing at least 100 OPQT, 10 ORDI, and BTC for fees.

vout2: **OP_RETURN:** `{"p":"brc-20","op":"transfer","tick":"OPQT","amt":"100"}`

vout3: **Recipient:** A standard output to Alice's address.

vout4: **OP_RETURN:** `{"p":"brc-20","op":"transfer","tick":"ORDI","amt":"10"}`

vout5: **Recipient:** A standard output to Bob's address.

vout6: **Change:** A standard output returning remaining BTC to the sender.

Indexer Logic: The Simplicity Indexer processes this sequence atomically. It first validates and executes the transfer to Alice, then validates and executes the transfer to Bob. Both operations are bound to the fate of the parent Bitcoin transaction.

4.2 Atomic Deploy and Full-Supply Mint

This pattern allows a token creator to deploy a new token and mint its entire supply to a specific address in a single, front-run-proof transaction.

Scenario: A project wants to launch "NEW" with a total supply of 1,000,000 and immediately secure all tokens in a treasury address.

Transaction Output Structure:

vout1: Input(s): From the creator, containing BTC for fees.

vout2: **OP_RETURN:** `{"p":"brc-20","op":"deploy","tick":"NEW",...}`

vout3: **OP_RETURN:** `{"p":"brc-20","op":"mint","tick":"NEW",...}`

vout4: **Recipient:** The project's treasury address, which will receive all tokens.

vout5: **Change:** (Optional) A final change output.

Indexer Logic:

1. The indexer processes Output 0, validating the deploy operation and registering "NEW" in its database. It consumes Input 0 as the deploy's structural partner.

2. It then immediately processes Output 1. Because the deploy operation was successfully processed just moments before in the same atomic context, the mint for "NEW" is now valid.

3. The 1,000,000 tokens are credited to the treasury address specified in Output 2.

This atomic composition eliminates front-running risks that plague traditional two-transaction deployment patterns.

4.3 Two-Party Atomic Swap (PSBT)

This advanced use case demonstrates how the Universal protocol can facilitate trust-minimized, peer-to-peer trades without a centralized intermediary.

Scenario: Alice agrees to trade 100 "OPQT" for 10 "ORDI" from Bob. They construct a single transaction that requires both of their signatures to be valid.

Transaction & PSBT Signing Flow:

1. **Construction:** A single Partially Signed Bitcoin Transaction (PSBT) is created with inputs from both parties and outputs reflecting the swap.

2. **Alice's Action:** Alice adds her input and signs only that input. She then sends the partially-signed PSBT to Bob.

3. **Bob's Action:** Bob inspects the PSBT. He verifies that he is receiving 100 OPQT and that the transaction correctly sends 10 ORDI from his account. He then adds his input and applies his signature.

4. **Broadcast:** The transaction now has all required signatures and can be broadcast to the network.

Transaction Structure:

Input 0: Alice's UTXO.

Input 1: Bob's UTXO.

Output 0 (OP_RETURN): Alice's transfer instruction.

Output 1 (Recipient): Bob's address, receiving the 100 OPQT.

Output 2 (OP_RETURN): Bob's transfer instruction.

Output 3 (Recipient): Alice's address, receiving the 10 ORDI.

Indexer Logic: The Simplicity Indexer sees a valid, fully-signed Bitcoin transaction with multiple inputs. It processes the sequenced transfers as normal, debiting and crediting each party according to the OP_RETURN instructions. The swap's atomicity is cryptographically guaranteed by Bitcoin itself—the entire transaction either succeeds or fails, ensuring neither party can be cheated.

5 Operation Proposal Improvements (OPIs)

The Universal protocol is designed for structured evolution through a community-driven governance process known as Operation Proposal Improvements (OPIs). Inspired by established standards-track processes like Bitcoin Improvement Proposals (BIPs), OPIs provide a formal framework for proposing, debating, and integrating new features into the protocol.

An OPI defines a new operation (`op`) or extends the interpretation of an existing one. This ensures the Universal protocol can adapt to new use cases without sacrificing its core principles of simplicity and security.

5.1 Purpose and Vision

The OPI process is designed to enable a rich ecosystem of functionalities directly on Bitcoin's base layer, including:

- **BRC-20 Harmonization:** Standardizing interactions and bridges with other token protocols (e.g., OPI-000: `no_return`).
- **Advanced Token Mechanics:** Enabling features such as token burning, time-locked releases, and conditional transfers.
- **Protocol Extensions:** Adding support for new operation types while maintaining indexer consensus.

5.2 OPI Lifecycle

Proposing and ratifying an OPI follows a transparent, community-vetted lifecycle:

1. **Drafting:** An author forks the OPI repository, copies the template, and writes a detailed proposal.
2. **Submission:** The proposal is submitted as a pull request to the official OPI repository.
3. **Community Review:** The proposal undergoes public discussion and technical review from the community and core developers.
4. **Ratification:** Once consensus is reached, the OPI is approved, assigned a final number, and merged.
5. **Implementation:** The ratified OPI is integrated as a new set of consensus rules into the Simplicity Indexer, making it a live feature of the Universal protocol.

5.3 OPI-000: no_return

OPI-000 introduces the `no_return` operation, a standardized mechanism for migrating BRC-20 tokens from the Ordinals inscription standard to the Universal protocol.

This process functions as a one-way bridge. A user burns their Ordinal-based transfer inscription by sending it to Satoshi's Genesis address. The `no_return` operation in the same transaction signals this intent to the Simplicity Indexer, which then credits the equivalent token amount to the user's Universal protocol balance.

no_return Transaction Structure:

Component	**Description**
Input #0	The user's UTXO containing a valid Ordinals BRC-20 transfer inscription.
Output #0	An OP_RETURN containing the `no_return` annotation.
Output #1	A standard output sending the Ordinal inscription to Satoshi's Genesis address, thereby burning it.

Table 4: OPI-000 no_return Transaction Structure

Upon validating a `no_return` transaction, the Simplicity Indexer reads the `tick` and `amt` from the Ordinal inscription and credits the Universal BRC-20 balance of the address that owned and spent Input #0. This creates a secure, verifiable, and one-way bridge from the Ordinals ledger to the Universal ledger.

Formal Specification:

Let I_0 denote the inscription UTXO, A_s the sender's address controlling I_0, and $(tick, amt)$ the token data encoded in the Ordinal inscription. The `no_return` operation performs:

$$\text{Balance}_{\text{Universal}}(A_s, tick) \leftarrow \text{Balance}_{\text{Universal}}(A_s, tick) + amt \tag{1}$$

provided that:

- Output #1 sends I_0 to `1A1zP1eP5QGefi2DMPTfTL5SLmv7DivfNa` (Genesis address)
- The inscription is valid under Ordinals consensus rules
- The operation is irreversible

6 Consensus Rules

The Simplicity Indexer MUST enforce the following rules to maintain a consistent global state. Any transaction violating these rules will be disregarded at the protocol layer.

1. **Operation Binding:** Each OP_RETURN operation is immutably bound to the first valid, non-OP_RETURN output that immediately follows it. The address of this output is the operation's recipient.
2. **Payload Integrity:** The OP_RETURN payload must be a valid, minified JSON object starting with `{"p":"brc-20"}`.
3. **Size Constraint:** The JSON payload must not exceed the 80-byte OP_RETURN data limit.
4. **Namespace Uniqueness:** A deploy operation is valid only if its `tick` has not been previously claimed on any valid BRC-20 standard (Ordinals or Universal).
5. **Flexible Keys:** The indexer must recognize `"max"` as an alias for `"m"` and `"lim"` as an alias for `"l"` in deploy operations.
6. **Token Existence:** `mint` and `transfer` operations must reference a validly deployed `tick`.
7. **Supply & Limits:** Operations must not violate the `max` supply or per-operation `lim` defined at deployment.
8. **Sufficient Balance:** A transfer is valid only if the sender's address, as identified by the input UTXOs, holds a sufficient balance according to the indexer's state.
9. **Dust Limit:** All standard Bitcoin outputs intended as recipients must meet the network's dust limit.

6.1 Formal State Transition Rules

Let S denote the global indexer state, T a Bitcoin transaction, and $\mathcal{O}_i$ the i-th OP_RETURN operation in T. The state transition function $\delta : S \times T \to S'$ is defined as:

$$S' = \delta(S,T) = \bigcirc_{i=1}^{n} \text{apply}(S, \mathcal{O}_i, T) \tag{2}$$

where $\bigcirc$ denotes sequential composition and apply validates each operation against the consensus rules before updating balances.

For a transfer operation $\mathcal{O}_{\text{transfer}}$ with parameters $(tick, amt)$:

$$\text{apply}(S, \mathcal{O}_{\text{transfer}}, T) = \begin{cases} S' & \text{if Balance}_S(A_{\text{sender}}, tick) \geq amt \\ S & \text{otherwise} \end{cases} \tag{3}$$

where S' is computed as:

$$\text{Balance}_{S'}(A_{\text{sender}}, tick) = \text{Balance}_S(A_{\text{sender}}, tick) - amt \tag{4}$$

$$\text{Balance}_{S'}(A_{\text{recipient}}, tick) = \text{Balance}_S(A_{\text{recipient}}, tick) + amt \tag{5}$$

7 Security Considerations

The Universal Protocol's security model relies on several critical assumptions and design choices that must be understood by implementers and users.

7.1 Indexer Reliance

The Universal protocol state is an interpretation of on-chain data maintained by Simplicity-compliant indexers. The integrity of the system relies on the correctness and consistency of its indexers. This introduces several considerations:

- **Determinism Requirement:** All compliant indexers must produce identical state when processing the same blockchain data.
- **Fork Risk:** If indexers diverge in their interpretation of consensus rules, the protocol can experience state forks, leading to double-spend scenarios in the token layer.
- **Mitigation:** The protocol specifies explicit, unambiguous consensus rules and provides a reference implementation (Simplicity Indexer) to minimize interpretation variance.

7.2 Critical Output Ordering

This is the most critical security consideration. The validity and destination of a token transfer are entirely dependent on the correct ordering of outputs. An OP_RETURN followed by the wrong output will result in tokens being sent to an unintended recipient.

Attack Vector Example:

```
Output 0: OP_RETURN {"p":"brc-20","op":"transfer","tick":"OPQT","amt
   ":"1000"}
Output 1: Attacker's address (546 sats)
Output 2: Intended recipient's address (546 sats)
```

In this malformed transaction, the 1,000 OPQT tokens would be credited to the attacker's address (Output 1) rather than the intended recipient (Output 2), because the protocol binds each OP_RETURN to the *immediately following* non-OP_RETURN output.

Mitigation Requirements:

- Wallet software MUST validate output ordering before signing
- Users MUST decode and inspect raw transactions before broadcast
- UI/UX MUST clearly display recipient addresses for each operation

7.3 Irreversible Operations

All valid Universal operations confirmed on the Bitcoin timechain are final. There is no mechanism to reverse a transaction sent to the wrong address or with an incorrect amount. This immutability, while consistent with Bitcoin's design philosophy, places the burden of correctness on transaction construction.

7.4 Namespace Squatting

The shared namespace with Ordinals BRC-20 means that token tickers are claimed on a first-come, first-served basis.

8 Best Practices & Optimization

8.1 Protection Against Accidental Burns

The Universal protocol's account-based model is its strongest safety feature. Balances are tied to addresses, not UTXOs. You cannot accidentally destroy your tokens by spending a UTXO for a normal Bitcoin transaction. As long as you control the private keys to an address, you retain full control of its tokens. A valid transfer operation is always required to move them.

This represents a significant safety improvement over UTXO-bound token protocols, where users have inadvertently destroyed valuable assets by spending UTXOs containing tokens without proper awareness.

8.2 Leverage Batching

For any scenario involving multiple recipients (e.g., exchange withdrawals, airdrops), use the multi-operation feature to dramatically reduce transaction fees and network load.

Cost Analysis:

Consider distributing tokens to n recipients. Under traditional single-operation transactions:

$$\text{Total Cost} = n \times (\text{base\_fee} + \text{input\_cost} + \text{output\_cost}) \tag{6}$$

Under multi-operation batching:

$$\text{Total Cost} = \text{base\_fee} + \text{input\_cost} + n \times (2 \times \text{output\_cost}) \tag{7}$$

The savings scale linearly with n, making batching essential for high-volume operations.

8.3 Minimize Payload Size

When deploying a token, use the shorter JSON keys (`m`, `l`) and a concise ticker to conserve bytes. This can be critical for fitting complex operations within the 80-byte limit.

Example Optimization:

```
// Verbose (47 bytes)
{"p":"brc-20","op":"deploy","tick":"OPQT","max":"21000000","lim
    ":"1000"}

// Optimized (43 bytes)
{"p":"brc-20","op":"deploy","tick":"OPQT","m":"21000000","l":"1000"}
```

8.4 Validate Before Signing

Always use a tool to decode and inspect the raw transaction before signing. Programmatically and manually verify that the OP_RETURN payloads and output ordering are exactly as intended.

Recommended Validation Checklist:

1. Decode all OP_RETURN outputs and verify JSON syntax
2. Confirm each operation's recipient address matches expectations
3. Verify output ordering (OP_RETURN immediately before recipient)
4. Check that amounts do not exceed available balances
5. Confirm dust limits are met for all recipient outputs

9 Comparative Analysis: Universal vs. Ordinals

Criterion	Universal Protocol (OP_RETURN)	Ordinals Inscription Standard
Data Location	Compact JSON in OP_RETURN outputs	Data inscribed in SegWit witness scripts
State Tracking	Off-chain by Simplicity Indexer	Off-chain by an Ordinals-aware indexer
Multi-Recipient	Native support via sequenced OP_RETURNs	Complex; requires multiple, separate transfers
On-chain Footprint	Prunable; extremely efficient per operation	Larger due to witness data and script overhead
Transaction Cost	Significantly lower, especially for batch operations	Higher
Logic	Explicit: operation and destination are linked pairs	Implicit: logic is tied to the satoshi itself
Safety Model	Account-based; tokens cannot be accidentally lost	UTXO-based; requires careful UTXO management
Extensibility	Structured (OPI process)	Ad-hoc; lacks formal governance

Table 5: Comparative Analysis of Universal and Ordinals BRC-20 Standards

9.1 Efficiency Analysis

Consider a transaction transferring tokens to 10 recipients:

Ordinals Approach:

- 10 separate transactions required
- Each transaction: witness data (∼200 bytes) + standard outputs
- Total size: ∼2,000 bytes of witness data
- Total fees: 10 × base_tx_fee

Universal Approach:

- 1 transaction with 10 OP_RETURN/recipient pairs
- Each operation: OP_RETURN (∼50 bytes) + recipient output
- Total size: ∼500 bytes of OP_RETURN data (prunable)
- Total fees: 1 × base_tx_fee

The Universal Protocol achieves approximately 75% fee reduction and 60% size reduction for this use case, with the added benefit of prunable data.

10 Reference Implementation: The Simplicity Indexer

The Simplicity Indexer is the canonical reference implementation for validating and tracking Universal Protocol operations. It functions as a deterministic state machine that processes Bitcoin blocks sequentially, maintaining a global token ledger.

10.1 Architecture

The indexer architecture consists of four primary components:

1. **Block Processor:** Fetches and validates Bitcoin blocks from a full node, extracting relevant transactions.
2. **Transaction Parser:** Decodes OP_RETURN outputs, extracts JSON payloads, and identifies operation types.
3. **State Machine:** Applies consensus rules to validate operations and update the token ledger.
4. **API Layer:** Provides REST/WebSocket interfaces for querying balances, transaction history, and protocol state.

10.2 State Machine Specification

The core state machine maintains the following data structures:

```
class SimplicitState:
    # Token registry: tick -> TokenMetadata
    tokens: Dict[str, TokenMetadata]

    # Balance ledger: (address, tick) -> amount
    balances: Dict[Tuple[str, str], Decimal]

    # Transaction history for indexing
    tx_history: List[Transaction]

    # Current block height
    block_height: int
```

10.3 Operation Processing Algorithm

```python
def process_transaction(tx: Transaction, state: SimplicitState) ->
    SimplicitState:
    """
␣␣␣␣Process␣a␣Universal␣Protocol␣transaction␣and␣update␣state.
␣␣␣␣Returns␣updated␣state␣or␣original␣state␣if␣transaction␣is␣invalid.
␣␣␣␣"""
    new_state = state.copy()

    for i, output in enumerate(tx.outputs):
        if not output.is_op_return():
            continue

        # Parse operation
        try:
            op = parse_op_return(output.script)
        except JSONDecodeError:
            continue  # Invalid JSON, skip

        # Validate protocol identifier
        if op.get('p') != 'brc-20':
            continue

        # Find recipient (next non-OP_RETURN output)
        recipient_output = find_next_recipient(tx.outputs, i)
        if not recipient_output:
            continue  # No valid recipient

        # Apply operation based on type
        if op['op'] == 'deploy':
            new_state = apply_deploy(op, recipient_output, new_state)
        elif op['op'] == 'mint':
            new_state = apply_mint(op, recipient_output, new_state)
        elif op['op'] == 'transfer':
            sender = identify_sender(tx.inputs)
            new_state = apply_transfer(op, sender, recipient_output,
                new_state)

    return new_state
```

10.4 Consensus Validation

Each operation processor implements strict consensus validation:

```python
def apply_transfer(op: Dict, sender: str, recipient: Output,
                   state: SimplicitState) -> SimplicitState:
    """
␣␣␣␣Apply␣a␣transfer␣operation␣with␣full␣consensus␣validation.
␣␣␣␣"""
    tick = op[’tick’]
    amt = Decimal(op[’amt’])
    recipient_addr = recipient.address

    # Consensus Rule: Token must exist
    if tick not in state.tokens:
        return state  # Invalid: token not deployed

    # Consensus Rule: Sufficient balance
    sender_balance = state.balances.get((sender, tick), Decimal(’0’))
    if sender_balance < amt:
        return state  # Invalid: insufficient balance

    # Consensus Rule: Dust limit
    if recipient.value < DUST_LIMIT:
        return state  # Invalid: output below dust

    # Valid operation: update balances
    new_state = state.copy()
    new_state.balances[(sender, tick)] = sender_balance - amt
    new_state.balances[(recipient_addr, tick)] = \
        state.balances.get((recipient_addr, tick), Decimal(’0’)) + amt

    return new_state
```

10.5 Pruning and Efficiency

Because OP_RETURN outputs are prunable, the Simplicity Indexer can operate efficiently without requiring the full blockchain history. The indexer only needs:

- Current UTXO set (for validating inputs)
- OP_RETURN data from relevant transactions (which can be stored in a compact database)
- Current token balances (maintained in the state machine)

This enables lightweight indexing infrastructure compared to full witness data storage required by Ordinals indexers.

11 Future Work

Several areas merit further research and development:

1. **Cross-Chain Bridges:** Formal specifications for trustless bridges to other chains, extending OPI-000's burn-and-mint pattern.
2. **Privacy Extensions:** Integration with confidential transaction schemes to enable private token transfers while maintaining protocol auditability.
3. **Layer 2 Integration:** Standards for representing Universal Protocol tokens in Lightning Network channels and other Layer 2 systems.
4. **Formal Verification:** Machine-checked proofs of protocol properties (e.g., no double-spending, balance conservation) using tools like Coq or Isabelle.
5. **Decentralized Indexer Network:** Economic incentive structures for running consensus-compatible indexers, potentially using proof-of-stake or proof-of-work mechanisms.
6. **Advanced Token Mechanics:** OPIs for features like time-locked vesting, multi-signature controlled tokens, and programmable transfer conditions.

12 Conclusion

The Universal Protocol represents a principled evolution of Bitcoin token infrastructure, addressing critical limitations in the original BRC-20 standard while preserving its accessibility and simplicity. By migrating operational logic to OP_RETURN outputs, the protocol achieves a "Satoshi-faithful" design that prioritizes long-term network health through prunable data structures.

The account-based safety model eliminates accidental token loss, while native support for atomic multi-operation transactions enables trust-minimized swaps and significant fee reduction. Through the OPI governance framework, the protocol provides a structured path for community-driven evolution without sacrificing consistency or introducing breaking changes.

The Simplicity Indexer serves as a deterministic reference implementation, ensuring that all compliant indexers converge on identical protocol state. Together, these components establish the most robust accountability system for fungible tokens on Bitcoin, explicitly linking intent, action, and outcome on-chain.

As the Bitcoin ecosystem continues to mature, the Universal Protocol provides a solid foundation for scalable token operations, honoring the radical simplicity that drove BRC-20's adoption while resolving its foundational flaws. We invite the community to build upon this foundation—block by block.

References

[1] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," Cryptography Mailing List, 2008.

[2] C. Rodarmor, "Ordinals: A numbering scheme for tracking individual satoshis," Ordinals Documentation, 2023.

[3] Domo, "BRC-20: An experimental fungible token standard for Bitcoin," 2023.

[4] G. Andresen, "OP_RETURN and Data in the Blockchain," Bitcoin Development Mailing List, 2012.

[5] A. Chow, "BIP 174: Partially Signed Bitcoin Transaction Format," Bitcoin Improvement Proposals, 2017.

[6] P. Wuille, et al., "BIP 341: Taproot: SegWit version 1 spending rules," Bitcoin Improvement Proposals, 2020.

[7] G. Maxwell, "CoinJoin: Bitcoin privacy for the real world," Bitcoin Forum, 2013.

[8] A. Back, "Hashcash - A Denial of Service Counter-Measure," 2002.