Resolution of Transaction Broadcast Failures in bit Library

Date: January 16, 2026

Writer : Salim Ben Khadija

Subject: Diagnosis and Remediation of Critical Failure in NetworkAPI Broadcast Logic

Component: bit Library (v0.8.0) | services.py

Scope: Bitcoin Mainnet & Testnet

1. Executive Summary

Users of the bit library encountered a critical "silent failure" when attempting to broadcast transactions using the standard .send() interface. While the transaction signing process was successful, the broadcast step failed to propagate the transaction to the network, despite often returning a transaction hash or failing silently without raising an exception.

An in-depth investigation isolated the root cause to the NetworkAPI layer. Specifically, the third-party service list contained deprecated providers ("bit rot"), and the only viable provider (BlockstreamAPI) contained a payload formatting bug that caused its requests to be rejected.

The issue was resolved by patching bit/network/services.py to correct the HTTP payload format for Blockstream and purging the dispatcher list of defunct services. This report details the full diagnostic path and the applied fixes for both network environments.

2. Incident Description

2.1 The Symptom

The failure manifested during the execution of the high-level key.send() method.

- **Observation:** The method would either return a transaction hash (implying success) or cycle through retries until raising a generic ConnectionError.
- **Verification:** Checking the returned hash on a block explorer revealed "Transaction Not Found." The transaction was never actually accepted by the network mempool.

2.2 Component Isolation

To diagnose the failure, the atomic operation was decomposed into two distinct stages:

1. **Stage 1: Signing (create_transaction)**
    - **Status:** Functional.
    - **Evidence:** The method produced a valid, cryptographically correct raw hex string.
2. **Stage 2: Broadcasting (NetworkAPI.broadcast_tx)**
    - **Status:** FAILED.
    - **Evidence:** Manual attempts to push the hex using the library's internal logic failed, while pushing the *same* hex via an external tool (e.g., cURL to Blockstream) succeeded. This confirmed the fault lay exclusively within the library's networking logic.

3. Root Cause Analysis (RCA)

The investigation revealed two concurrent failures within the bit/network/services.py module.

3.1 RC-1: Deprecated Service Dependencies (System Entropy)

The library's NetworkAPI dispatcher relies on a hardcoded list of third-party APIs. Due to the library's age, this list had degraded significantly:

- **Smartbit:** Service has been permanently shut down.
- **Bitcore:** API endpoints are unreliable or unreachable.
- **Blockchair:** The library lacks the necessary API Key integration required by current endpoint security policies.

3.2 RC-2: Malformed Payload in BlockstreamAPI (The Critical Bug)

The only operational service provider in the list was **Blockstream**. However, the library's wrapper class (BlockstreamAPI) was constructing the HTTP request incorrectly for the modern API specification.

- **Expected Format:** The Blockstream /tx endpoint expects the **raw hex string** as the HTTP POST body.
- **Actual Implementation:** The library was wrapping the hex in a form-encoded dictionary: data={'data': <hex_string>}.
- **Outcome:** The Blockstream server rejected the request (likely 400 Bad Request), but the library's "round-robin" logic interpreted this as a generic connection failure and moved on to the dead services (RC-1), eventually causing the entire operation to fail.

This bug was present in **both** Mainnet (broadcast_tx) and Testnet (broadcast_tx_testnet) methods.

4. Remediation and Corrective Actions

The resolution involved "surgical" modifications to bit/network/services.py. No external dependencies were introduced; the fix focused on correcting the existing logic.

4.1 Fix 1: Correcting the HTTP Payload (Protocol Repair)

We modified the BlockstreamAPI class to send the transaction hex directly as the request body, removing the dictionary wrapper.

File: bit/network/services.py

Class: BlockstreamAPI

A. Mainnet Fix (broadcast_tx)

```
# [BEFORE] Broken Implementation
@classmethod
def broadcast_tx(cls, tx_hex):
    # Incorrect: Sends data={'data': '01000...'}
    r = requests.post(cls.MAIN_TX_PUSH_API, data={cls.TX_PUSH_PARAM: tx_hex},
timeout=DEFAULT_TIMEOUT)
    return True if r.status_code == 200 else False

# [AFTER] Fixed Implementation
@classmethod
def broadcast_tx(cls, tx_hex):
    # Correct: Sends raw hex string '01000...'
    r = requests.post(cls.MAIN_TX_PUSH_API, data=tx_hex, timeout=DEFAULT_TIMEOUT)
    return True if r.status_code == 200 else False
```

B. Testnet Fix (broadcast_tx_testnet)

```
# [BEFORE] Broken Implementation
@classmethod
def broadcast_tx_testnet(cls, tx_hex):
    r = requests.post(cls.TEST_TX_PUSH_API, data={cls.TX_PUSH_PARAM: tx_hex},
timeout=DEFAULT_TIMEOUT)
    return True if r.status_code == 200 else False

# [AFTER] Fixed Implementation
@classmethod
def broadcast_tx_testnet(cls, tx_hex):
    r = requests.post(cls.TEST_TX_PUSH_API, data=tx_hex, timeout=DEFAULT_TIMEOUT)
    return True if r.status_code == 200 else False
```

4.2 Fix 2: Purging Defunct Services (Dispatcher Cleanup)

To prevent latency and false-negative errors, we removed the broken providers from the NetworkAPI configuration lists. This forces the library to use the single, known-good path (Blockstream).

File: bit/network/services.py

Class: NetworkAPI

A. Mainnet Configuration (BROADCAST_TX_MAIN)

```
# [BEFORE] Contains broken services
BROADCAST_TX_MAIN = [
    BlockchairAPI.broadcast_tx,
    BlockstreamAPI.broadcast_tx,
    BitcoreAPI.broadcast_tx,
    SmartbitAPI.broadcast_tx,
    BlockchainAPI.broadcast_tx,
]

# [AFTER] Purged List
BROADCAST_TX_MAIN = [
    BlockstreamAPI.broadcast_tx,  # Sole reliable provider
]
```

B. Testnet Configuration (BROADCAST_TX_TEST)

```
# [BEFORE] Contains broken services
BROADCAST_TX_TEST = [
    BlockchairAPI.broadcast_tx_testnet,
    BlockstreamAPI.broadcast_tx_testnet,
    BitcoreAPI.broadcast_tx_testnet,
    SmartbitAPI.broadcast_tx_testnet,
]

# [AFTER] Purged List
BROADCAST_TX_TEST = [
    BlockstreamAPI.broadcast_tx_testnet,  # Sole reliable provider
]
```

5. Verification Results

Following the application of the patch, the standard .send() method was re-evaluated on both networks.

Test Case	Method Used	Result	Verification
Testnet	k.send(...)	**SUCCESS**	Hash confirmed on blockstream.info/testnet
Mainnet	k.send(...)	**SUCCESS**	Hash confirmed on blockstream.info

Conclusion

The breakdown of the bit library's broadcasting capability was caused by outdated service definitions and a specific protocol mismatch in the BlockstreamAPI implementation. By correcting the payload structure and isolating the reliable network provider, we have fully restored the library's native functionality without the need for external wrapper scripts or additional dependencies.

The fix is robust, maintainable, and strictly adheres to the library's original architectural scope.