```
BIP: 64
Layer: Peer Services
Title: getutxo message
Author: Mike Hearn <hearn@vinumeris.com>
Comments-Summary: No comments yet.
Comments-URI: https://github.com/bitcoin/bips/wiki/Comments:BIP-0064
Status: Obsolete
Type: Standards Track
Created: 2014-06-10
```

Abstract

This document describes a small P2P protocol extension that performs UTXO lookups given a set of outpoints.

Motivation

All full Bitcoin nodes maintain a database called the unspent transaction output set. This set is how double spending is checked for: to be valid a transaction must identify unspent outputs in this set using an identifier called an "outpoint", which is merely the hash of the output's containing transaction plus an index.

The ability to query this can sometimes be useful for a lightweight/SPV client which does not have the full UTXO set at hand. For example, it can be useful in applications implementing assurance contracts to do a quick check when a new pledge becomes visible to test whether that pledge was already revoked via a double spend. Although this message is not strictly necessary because e.g. such an app could be implemented by fully downloading and storing the block chain, it is useful for obtaining acceptable performance and resolving various UI cases.

Another example of when this data can be useful is for performing floating fee calculations in an SPV wallet. This use case requires some other changes to the Bitcoin protocol however, so we will not dwell on it here.

Specification

Two new messages are defined. The "getutxos" message has the following structure:

Field Size	Description	Data type	Comments
1	check mempool	bool	Whether to apply mempool transactions during the calculation, th
?	outpoints	vector	The list of outpoints to be queried. Each outpoint is serialized in t

The response message "utxos" has the following structure:

Field Size	Description	Data type	Comments
4	chain height	uint32	The height of the chain at the moment the result was calculated.
32	chain tip hash	uint256	Block hash of the top of the chain at the moment the result was cal
?	hit bitmap	byte[]	An array of bytes encoding one bit for each outpoint queried. Each
?	result utxos	result[]	A list of result objects (defined below), one for each outpoint that i

The result object is defined as:

Field Size	Description	Data type	Comments
4	tx version	uint32	The version number of the transaction the UTXO was found in.
4	height	uint32	The height of the block containing the defining transaction, or 0x7FF
?	output	CTxOut	The output itself, serialized in the same way as in a tx message.

Backward compatibility

Nodes indicate support by advertising a protocol version above 70003 and by setting a new NODE_GETUTXO flag in their nServices field, which has a value of 2 (the second bit of the field).

Authentication

The UTXO set is not currently authenticated by anything. There are proposals to resolve this by introducing a new consensus rule that commits to a root hash of the UTXO set in blocks, however this feature is not presently available in the Bitcoin protocol. Once it is, the utxos message could be upgraded to include Merkle branches showing inclusion of the UTXOs in the committed sets.

If the requesting client is looking up outputs for a signed transaction that they have locally, the client can partly verify the returned output by running the input scripts with it. Currently this verifies only that the script is correct. A future version of the Bitcoin protocol is likely to also allow the value to be checked in this way. It does not show that the output is really unspent or was ever actually created in the block chain however. Additionally, the form of the provided scriptPubKey should be checked before execution to ensure the remote peer doesn't just set the script to OP_TRUE.

If the requesting client has a mapping of chain heights to block hashes in the best chain e.g. obtained via getheaders, then they can obtain a proof that the output did at one point exist by requesting the block and searching for the output within it. When combined with Bloom filtering this can be reasonably efficient.

Note that even when the outputs are being checked against something this protocol has the same security model as Bloom filtering: a remote node can lie through omission by claiming the requested UTXO does not exist / was already spent (they are the same, from the perspective of a full node). Querying multiple nodes and combining their answers can be a partial solution to this, although as nothing authenticates the Bitcoin P2P network a man in the middle could still yield incorrect results.

Implementation

https://github.com/bitcoin/bitcoin/pull/4351/files