```
BIP: 61
Layer: Peer Services
Title: Reject P2P message
Author: Gavin Andresen <gavinandresen@gmail.com>
Comments-Summary: Controversial; some recommendation, and some discouragement
Comments-URI: https://github.com/bitcoin/bips/wiki/Comments:BIP-0061
Status: Final
Type: Standards Track
Created: 2014-06-18
```

Abstract

This BIP describes a new message type for the Bitcoin peer-to-peer network.

Motivation

Giving peers feedback about why their blocks or transactions are rejected, or why they are being banned for not following the protocol helps interoperability between different implementations.

It also gives SPV (simplified payment verification) clients a hint that something may be wrong when their transactions are rejected due to insufficient priority or fees.

Specification

Data types in this specification are as described at https://en.bitcoin.it/wiki/Protocol_specification

reject

One new message type "reject" is introduced. It is sent directly to a peer in response to a "version", "tx" or "block" message.

For example, the message flow between two peers for a relayed transaction that is rejected for some reason would be:

```
--> inv
<-- getdata
--> tx
<-- reject
```

All implementations of the P2P protocol version 70,002 and later should support the reject message.

common payload Every reject message begins with the following fields. Some messages append extra, message-specific data.

Field Size	Name	Data type	Comments
variable	response-to-msg	var_str	Message that triggered the reject
1	reject-code	uint8_t	0x01 through 0x4f (see below)
variable	reason	var_string	Human-readable message for debugging

The human-readable string is intended only for debugging purposes; in particular, different implementations may use different strings. The string should not be shown to users or used for anthing besides diagnosing interoperability problems.

The following reject code categories are used; in the descriptions below, "server" is the peer generating the reject message, "client" is the peer that will receive the message.

Range	Category
0x01-0x0f	Protocol syntax errors
0x10-0x1f	Protocol semantic errors
0x40-0x4f	Server policy rule

rejection codes common to all message types

Code	Description
0x01	Message could not be decoded

reject version codes Codes generated during the initial connection process in response to a "version" message:

Code	Description
0x11	Client is an obsolete, unsupported version
0x12	Duplicate version message received

reject tx payload, codes Transaction rejection messages extend the basic message with the transaction id hash:

Field Size	Name	Data type	Comments
32	hash	char[32]	transaction that is rejected

The following codes are used:

Code	Description

0x10	Transaction is invalid for some reason (invalid signature, output value greater than input, etc.)
0x12	An input is already spent
0x40	Not mined/relayed because it is "non-standard" (type or version unknown by the server)
0x41	One or more output amounts are below the 'dust' threshold
0x42	Transaction does not have enough fee/priority to be relayed or mined

payload, reject block Block rejection messages extend the basic message with the block header hash:

Field Size	Name	Data type	Comments
32	hash	char[32]	block (hash of block header) that is rejected

Rejection codes:

code	description
0x10	Block is invalid for some reason (invalid proof-of-work, invalid signature, etc)
0x11	Block's version is no longer supported
0x43	Inconsistent with a compiled-in checkpoint

Note: blocks that are not part of the server's idea of the current best chain, but are otherwise valid, should not trigger reject messages.

Compatibility

The reject message is backwards-compatible; older peers that do not recognize the reject message will ignore it.

Implementation notes

Implementors must consider what happens if an attacker either sends them reject messages for valid transactions/blocks or sends them random reject messages, and should beware of possible denial-of-service attacks. For example, notifying the user of every reject message received would make it trivial for an attacker to mount an annoy-the-user attack. Even merely writing every reject message to a debugging log could make an implementation vulnerable to a fill-up-the-users-disk attack.