```
BIP: 155
Layer: Peer Services
Title: addrv2 message
Author: Wladimir J. van der Laan <laanwj@gmail.com>
Comments-Summary: No comments yet.
Comments-URI: https://github.com/bitcoin/bips/wiki/Comments:BIP-0155
Status: Draft
Type: Standards Track
Created: 2019-02-27
License: BSD-2-Clause
```

Introduction

Abstract

This document proposes a new P2P message to gossip longer node addresses over the P2P network. This is required to support new-generation Onion addresses, I2P, and potentially other networks that have longer endpoint addresses than fit in the 128 bits of the current `addr` message.

Copyright

Motivation

Tor v3 hidden services are part of the stable release of Tor since version 0.3.2.9. They have various advantages compared to the old hidden services, among which better encryption and privacy [1]. These services have 256 bit addresses and thus do not fit in the existing `addr` message, which encapsulates onion addresses in OnionCat IPv6 addresses.

Other transport-layer protocols such as I2P have always used longer addresses. This change would make it possible to gossip such addresses over the P2P network, so that other peers can connect to them.

Specification

> The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119[2].

The `addrv2` message is defined as a message where `pchCommand == "addrv2"`. It is serialized in the standard encoding for P2P messages. Its format is similar to the current `addr` message format, with the difference that the fixed 16-byte

[1] Tor Rendezvous Specification - Version 3
[2] RFC 2119

IP address is replaced by a network ID and a variable-length address, and the services format has been changed to CompactSize.

This means that the message contains a serialized `std::vector` of the following structure:

Type	Name	Description
`uint32_t`	`time`	Time that this node was last seen as connected to the network. A time in Unix
`CompactSize`	`services`	Service bits. A bit field that is 64 bits wide, encoded in CompactSize.
`uint8_t`	`networkID`	Network identifier. An 8-bit value that specifies which network is addressed.
`std::vector`	`addr`	Network address. The interpretation depends on networkID.
`uint16_t`	`port`	Network port. If not relevant for the network this MUST be 0.

One message can contain up to 1,000 addresses. Clients SHOULD reject messages with more addresses.

Field `addr` has a variable length, with a maximum of 512 bytes (4096 bits). Clients SHOULD reject messages with longer addresses, irrespective of the network ID.

The list of reserved network IDs is as follows:

Network ID	Enumeration	Address length (bytes)	Description
`0x01`	`IPV4`	4	IPv4 address (globally routed internet)
`0x02`	`IPV6`	16	IPv6 address (globally routed internet)
`0x03`	`TORV2`	10	Tor v2 hidden service address
`0x04`	`TORV3`	32	Tor v3 hidden service address
`0x05`	`I2P`	32	I2P overlay network address
`0x06`	`CJDNS`	16	Cjdns overlay network address

Clients are RECOMMENDED to gossip addresses from all known networks even if they are currently not connected to some of them. That could help multi-homed nodes and make it more difficult for an observer to tell which networks a node is connected to.

Clients SHOULD NOT gossip addresses from unknown networks because they have no means to validate those addresses and so can be tricked to gossip invalid addresses.

Further network ID numbers MUST be reserved in a new BIP document.

Clients SHOULD reject messages that contain addresses that have a different length than specified in this table for a specific network ID, as these are meaningless.

See the appendices for the address encodings to be used for the various networks.

Signaling support and compatibility

Introduce a new message type `sendaddrv2`. Sending such a message indicates that a node can understand and prefers to receive `addrv2` messages instead of `addr` messages. I.e. "Send me addrv2". Sending or not sending this message does not imply any preference with respect to receiving unrequested address messages.

The `sendaddrv2` message MUST only be sent in response to the `version` message from a peer and prior to sending the `verack` message.

For older peers, that did not emit `sendaddrv2`, keep sending the legacy `addr` message, ignoring addresses with the newly introduced address types.

Reference implementation

The reference implementation is available at (to be done)

Acknowledgements

- Jonas Schnelli: change `services` field to CompactSize, to make the message more compact in the likely case instead of always using 8 bytes.

- Gregory Maxwell: various suggestions regarding extensibility

Appendix A: Tor v2 address encoding

The new message introduces a separate network ID for `TORV2`.

Clients MUST send Tor hidden service addresses with this network ID, with the 80-bit hidden service ID in the address field. This is the same as the representation in the legacy `addr` message, minus the 6 byte prefix of the OnionCat wrapping.

Clients SHOULD ignore OnionCat (`fd87:d87e:eb43::/48`) addresses on receive if they come with the `IPV6` network ID.

Appendix B: Tor v3 address encoding

According to the spec [3], next-gen `.onion` addresses are encoded as follows:

```
onion_address = base32(PUBKEY | CHECKSUM | VERSION) + ".onion"
 CHECKSUM = H(".onion checksum" | PUBKEY | VERSION)[:2]

 where:
   - PUBKEY is the 32 bytes ed25519 master pubkey of the hidden service
   - VERSION is a one byte version field (default value '\x03')
   - ".onion checksum" is a constant string
   - CHECKSUM is truncated to two bytes before inserting it in onion_address
```

[3] Tor Rendezvous Specification - Version 3: Encoding onion addresses

```
   - H() is the SHA3-256 cryptographic hash function
```

Tor v3 addresses MUST be sent with the `TORV3` network ID, with the 32-byte PUBKEY part in the address field. As VERSION will always be '\x03' in the case of v3 addresses, this is enough to reconstruct the onion address.

Appendix C: I2P address encoding

Like Tor, I2P naming uses a base32-encoded address format[4].

I2P uses 52 characters (256 bits) to represent the full SHA-256 hash, followed by `.b32.i2p`.

I2P addresses MUST be sent with the `I2P` network ID, with the decoded SHA-256 hash as address field.

Appendix D: Cjdns address encoding

Cjdns addresses are simply IPv6 addresses in the `fc00::/8` range[5]. They MUST be sent with the `CJDNS` network ID.

References

[4]I2P: Naming and address book
[5]Cjdns whitepaper: Pulling It All Together