
py-libp2p Documentation

Release 0.1.5

The Ethereum Foundation

Apr 27, 2024

GENERAL

1	Contents	3
	Python Module Index	85
	Index	87

The Python implementation of the libp2p networking stack

CONTENTS

1.1 Introduction

py-libp2p is the Python implementation of the libp2p networking stack. It hopes to someday be fully compatible with other implementations of libp2p.

This project is still in its early stages and is not yet ready for production use.

1.1.1 Further Reading

- [libp2p main site](#)
- [Tutorial: Introduction to libp2p](#)

1.2 Quickstart

TODO - add quickstart instructions

1.3 Release Notes

1.3.1 libp2p v0.1.5 (2020-03-25)

Features

- Dial all multiaddrs stored for a peer when attempting to connect (not just the first one in the peer store). (#386)
- Migrate transport stack to trio-compatible code. Merge in #404. (#396)
- Migrate network stack to trio-compatible code. Merge in #404. (#397)
- Migrate host, peer and protocols stacks to trio-compatible code. Merge in #404. (#398)
- Migrate muxer and security transport stacks to trio-compatible code. Merge in #404. (#399)
- Migrate pubsub stack to trio-compatible code. Merge in #404. (#400)
- Fix interop tests w/ new trio-style code. Merge in #404. (#401)
- Fix remainder of test code w/ new trio-style code. Merge in #404. (#402)
- Add initial infrastructure for *noise* security transport. (#405)

- Add *PatternXX* of *noise* security transport. (#406)
- The *msg_id* in a pubsub message is now configurable by the user of the library. (#410)

Bugfixes

- Use *sha256* when calculating a peer's ID from their public key in Kademlia DHTs. (#385)
- Store peer ids in *set* instead of *list* and check if peer id exists in *dict* before accessing to prevent *KeyError*. (#387)
- Do not close a connection if it has been reset. (#394)

Internal Changes - for py-libp2p Contributors

- Add support for *fastecdsa* on windows (and thereby supporting windows installation via *pip*) (#380)
- Prefer f-string style formatting everywhere except logging statements. (#389)
- Mark *lru* dependency as third-party to fix a windows inconsistency. (#392)
- Bump *multiaddr* dependency to version *0.0.9* so that *multiaddr* objects are hashable. (#393)
- Remove incremental mode of *mypy* to disable some warnings. (#403)

1.3.2 libp2p v0.1.4 (2019-12-12)

Features

- Added support for Python 3.6 (#372)
- Add signing and verification to pubsub (#362)

Internal Changes - for py-libp2p Contributors

- Refactor and cleanup gossipsub (#373)

1.3.3 libp2p v0.1.3 (2019-11-27)

Bugfixes

- Handle *Stream** errors (like *StreamClosed*) during calls to *stream.write()* and *stream.read()* (#350)
- Relax the *protobuf* dependency to play nicely with other libraries. It was pinned to 3.9.0, and now permits v3.10 up to (but not including) v4. (#354)
- Fixes *KeyError* when peer in a stream accidentally closes and resets the stream, because handlers for both will try to *del streams[stream_id]* without checking if the entry still exists. (#355)

Improved Documentation

- Use Sphinx & autodoc to generate docs, now available on py-libp2p.readthedocs.io (#318)

Internal Changes - for py-libp2p Contributors

- Added Makefile target to test a packaged version of libp2p before release. (#353)
- Move helper tools from `tests/` to `libp2p/tools/`, and some mildly-related cleanups. (#356)

Miscellaneous changes

- #357

1.3.4 v0.1.2

Welcome to the great beyond, where changes were not tracked by release...

1.4 examples package

1.4.1 Subpackages

`examples.chat` package

Submodules

`examples.chat.chat` module

`examples.chat.chat.main()` → `None`

async `examples.chat.chat.read_data(stream: INetStream)` → `None`

async `examples.chat.chat.run(port: int, destination: str)` → `None`

async `examples.chat.chat.write_data(stream: INetStream)` → `None`

Module contents

1.4.2 Module contents

1.5 libp2p package

1.5.1 Subpackages

`libp2p.crypto` package

Subpackages

libp2p.crypto.pb package

Submodules

libp2p.crypto.pb.crypto_pb2 module

Generated protocol buffer code.

```
class libp2p.crypto.pb.crypto_pb2.PrivateKey
    Bases: Message, Message
    DESCRIPTOR = <google._upb._message.Descriptor object>

class libp2p.crypto.pb.crypto_pb2.PublicKey
    Bases: Message, Message
    DESCRIPTOR = <google._upb._message.Descriptor object>
```

Module contents

Submodules

libp2p.crypto.authenticated_encryption module

```
class libp2p.crypto.authenticated_encryption.EncryptionParameters(cipher_type: str, hash_type: str, iv: bytes, mac_key: bytes, cipher_key: bytes)

    Bases: object
    cipher_key: bytes
    cipher_type: str
    hash_type: str
    iv: bytes
    mac_key: bytes

exception libp2p.crypto.authenticated_encryption.InvalidMACException
    Bases: Exception

class libp2p.crypto.authenticated_encryption.MacAndCipher(parameters: EncryptionParameters)
    Bases: object
    authenticate(data: bytes) → bytes
    decrypt_if_valid(data_with_tag: bytes) → bytes
    encrypt(data: bytes) → bytes
```

```
libp2p.crypto.authenticated_encryption.initialize_pair(cipher_type: str, hash_type: str, secret:
                                                    bytes) → Tuple[EncryptionParameters,
                                                    EncryptionParameters]
```

Return a pair of Keys for use in securing a communications channel with authenticated encryption derived from the secret and using the requested `cipher_type` and `hash_type`.

libp2p.crypto.ecc module

```
class libp2p.crypto.ecc.ECCPrivateKey(impl: int, curve: <MagicMock name='mock.curve.Curve'
                                         id='140457861788048'>)
```

Bases: *PrivateKey*

`get_public_key()` → *PublicKey*

`get_type()` → *KeyType*

Returns the *KeyType* for `self`.

`classmethod new(curve: str)` → *ECCPrivateKey*

`sign(data: bytes)` → *bytes*

`to_bytes()` → *bytes*

Returns the byte representation of this key.

```
class libp2p.crypto.ecc.ECCPublicKey(impl: <MagicMock name='mock.point.Point'
                                         id='140457861800048'>, curve: <MagicMock
                                         name='mock.curve.Curve' id='140457861788048'>)
```

Bases: *PublicKey*

`classmethod from_bytes(data: bytes, curve: str)` → *ECCPublicKey*

`get_type()` → *KeyType*

Returns the *KeyType* for `self`.

`to_bytes()` → *bytes*

Returns the byte representation of this key.

`verify(data: bytes, signature: bytes)` → *bool*

Verify that `signature` is the cryptographic signature of the hash of `data`.

```
libp2p.crypto.ecc.create_new_key_pair(curve: str) → KeyPair
```

Return a new ECC keypair with the requested curve type, e.g. “P-256”.

```
libp2p.crypto.ecc.infer_local_type(curve: str) → <MagicMock name='mock.curve.Curve'
                                         id='140457861788048'>
```

Convert a `str` representation of some elliptic curve to a representation understood by the backend of this module.

libp2p.crypto.ed25519 module

```
class libp2p.crypto.ed25519.Ed25519PrivateKey(impl: PrivateKey)
    Bases: PrivateKey
    classmethod from_bytes(data: bytes) → Ed25519PrivateKey
    get_public_key() → PublicKey
    get_type() → KeyType
        Returns the KeyType for self.
    classmethod new(seed: bytes | None = None) → Ed25519PrivateKey
    sign(data: bytes) → bytes
    to_bytes() → bytes
        Returns the byte representation of this key.
class libp2p.crypto.ed25519.Ed25519PublicKey(impl: PublicKey)
    Bases: PublicKey
    classmethod from_bytes(key_bytes: bytes) → Ed25519PublicKey
    get_type() → KeyType
        Returns the KeyType for self.
    to_bytes() → bytes
        Returns the byte representation of this key.
    verify(data: bytes, signature: bytes) → bool
        Verify that signature is the cryptographic signature of the hash of data.
libp2p.crypto.ed25519.create_new_key_pair(seed: bytes | None = None) → KeyPair
```

libp2p.crypto.exceptions module

```
exception libp2p.crypto.exceptions.CryptographyError
    Bases: BaseLibp2pError
exception libp2p.crypto.exceptions.MissingDeserializerError
    Bases: CryptographyError
    Raise if the requested deserialization routine is missing for some type of cryptographic key.
```

libp2p.crypto.key_exchange module

```
libp2p.crypto.key_exchange.create_ephemeral_key_pair(curve_type: str) → Tuple[PublicKey,
                                                                              Callable[[bytes], bytes]]
    Facilitates ECDH key exchange.
```

libp2p.crypto.keys module**class** libp2p.crypto.keys.**Key**Bases: [ABC](#)

A Key represents a cryptographic key.

abstract **get_type()** → [KeyType](#)

Returns the KeyType for self.

abstract **to_bytes()** → [bytes](#)

Returns the byte representation of this key.

class libp2p.crypto.keys.**KeyPair**(*private_key*: libp2p.crypto.keys.PrivateKey, *public_key*: libp2p.crypto.keys.PublicKey)Bases: [object](#)**private_key**: [PrivateKey](#)**public_key**: [PublicKey](#)**class** libp2p.crypto.keys.**KeyType**(*value*)Bases: [Enum](#)

An enumeration.

ECC_P256 = 4**ECDSA** = 3**Ed25519** = 1**RSA** = 0**Sepc256k1** = 2**class** libp2p.crypto.keys.**PrivateKey**Bases: [Key](#)

A PrivateKey represents a cryptographic private key.

classmethod **deserialize_from_protobuf**(*protobuf_data*: [bytes](#)) → [PrivateKey](#)**abstract** **get_public_key()** → [PublicKey](#)**serialize()** → [bytes](#)

Return the canonical serialization of this Key.

abstract **sign**(*data*: [bytes](#)) → [bytes](#)**class** libp2p.crypto.keys.**PublicKey**Bases: [Key](#)

A PublicKey represents a cryptographic public key.

classmethod **deserialize_from_protobuf**(*protobuf_data*: [bytes](#)) → [PublicKey](#)**serialize()** → [bytes](#)

Return the canonical serialization of this Key.

abstract **verify**(*data*: [bytes](#), *signature*: [bytes](#)) → [bool](#)

Verify that signature is the cryptographic signature of the hash of data.

libp2p.crypto.rsa module

```
class libp2p.crypto.rsa.RSAPrivateKey(impl: RsaKey)
    Bases: PrivateKey
    get_public_key() → PublicKey
    get_type() → KeyType
        Returns the KeyType for self.
    classmethod new(bits: int = 2048, e: int = 65537) → RSAPrivateKey
    sign(data: bytes) → bytes
    to_bytes() → bytes
        Returns the byte representation of this key.
class libp2p.crypto.rsa.RSAPublicKey(impl: RsaKey)
    Bases: PublicKey
    classmethod from_bytes(key_bytes: bytes) → RSAPublicKey
    get_type() → KeyType
        Returns the KeyType for self.
    to_bytes() → bytes
        Returns the byte representation of this key.
    verify(data: bytes, signature: bytes) → bool
        Verify that signature is the cryptographic signature of the hash of data.
libp2p.crypto.rsa.create_new_key_pair(bits: int = 2048, e: int = 65537) → KeyPair
    Returns a new RSA keypair with the requested key size (bits) and the given public exponent e.
    Sane defaults are provided for both values.
```

libp2p.crypto.secp256k1 module

```
class libp2p.crypto.secp256k1.Secp256k1PrivateKey(impl: PrivateKey)
    Bases: PrivateKey
    classmethod deserialize(data: bytes) → Secp256k1PrivateKey
    classmethod from_bytes(data: bytes) → Secp256k1PrivateKey
    get_public_key() → PublicKey
    get_type() → KeyType
        Returns the KeyType for self.
    classmethod new(secret: bytes | None = None) → Secp256k1PrivateKey
    sign(data: bytes) → bytes
    to_bytes() → bytes
        Returns the byte representation of this key.
```

class libp2p.crypto.secp256k1.**Secp256k1PublicKey**(*impl: PublicKey*)

Bases: *PublicKey*

classmethod **deserialize**(*data: bytes*) → *Secp256k1PublicKey*

classmethod **from_bytes**(*data: bytes*) → *Secp256k1PublicKey*

get_type() → *KeyType*

Returns the KeyType for self.

to_bytes() → *bytes*

Returns the byte representation of this key.

verify(*data: bytes, signature: bytes*) → *bool*

Verify that *signature* is the cryptographic signature of the hash of *data*.

libp2p.crypto.secp256k1.**create_new_key_pair**(*secret: bytes | None = None*) → *KeyPair*

Returns a new Secp256k1 keypair derived from the provided *secret*, a sequence of bytes corresponding to some integer between 0 and the group order.

A valid secret is created if *None* is passed.

libp2p.crypto.serialization module

libp2p.crypto.serialization.**deserialize_private_key**(*data: bytes*) → *PrivateKey*

libp2p.crypto.serialization.**deserialize_public_key**(*data: bytes*) → *PublicKey*

Module contents

libp2p.host package

Submodules

libp2p.host.basic_host module

class libp2p.host.basic_host.**BasicHost**(*network: INetworkService, default_protocols: OrderedDict[TProtocol, StreamHandlerFn] = None*)

Bases: *IHost*

BasicHost is a wrapper of a *INetwork* implementation.

It performs protocol negotiation on a stream with multistream-select right after a stream is initialized.

async **close**() → *None*

async **connect**(*peer_info: PeerInfo*) → *None*

Ensure there is a connection between this host and the peer with given *peer_info.peer_id*. *connect* will absorb the addresses in *peer_info* into its internal peerstore. If there is not an active connection, *connect* will issue a dial, and block until a connection is opened, or an error is returned.

Parameters

peer_info (*peer.peerinfo.PeerInfo*) – *peer_info* of the peer we want to connect to

async disconnect(*peer_id*: ID) → None

get_addrs() → List[Multiaddr]

Returns

all the multiaddr addresses this host is listening to

get_id() → ID

Returns

peer_id of host

get_mux() → Multiselect

Returns

mux instance of host

get_network() → INetworkService

Returns

network instance of host

get_peerstore() → IPeerStore

Returns

peerstore of the host (same one as in its network instance)

get_private_key() → PrivateKey

Returns

the private key belonging to the peer

get_public_key() → PublicKey

Returns

the public key belonging to the peer

multiselect: Multiselect

multiselect_client: MultiselectClient

async new_stream(*peer_id*: ID, *protocol_ids*: Sequence[TProtocol]) → INetStream

Parameters

- **peer_id** – peer_id that host is connecting
- **protocol_ids** – available protocol ids to use for stream

Returns

stream: new stream created

peerstore: IPeerStore

run(*listen_addrs*: Sequence[Multiaddr]) → AsyncIterator[None]

Run the host instance and listen to `listen_addrs`.

Parameters

listen_addrs – a sequence of multiaddrs that we want to listen to

set_stream_handler(*protocol_id*: *TProtocol*, *stream_handler*: *Callable*[[*TNetStream*], *Awaitable*[*None*]])
→ *None*

Set stream handler for given *protocol_id*

Parameters

- **protocol_id** – protocol id used on stream
- **stream_handler** – a stream handler function

libp2p.host.defaults module

libp2p.host.defaults.get_default_protocols(*host*: *IHost*) → *OrderedDict*[*TProtocol*, *StreamHandlerFn*]

libp2p.host.exceptions module

exception libp2p.host.exceptions.ConnectionFailure

Bases: *HostException*

exception libp2p.host.exceptions.HostException

Bases: *BaseLibp2pError*

A generic exception in *IHost*.

exception libp2p.host.exceptions.StreamFailure

Bases: *HostException*

libp2p.host.host_interface module

class libp2p.host.host_interface.IHost

Bases: *ABC*

abstract async close() → *None*

abstract async connect(*peer_info*: *PeerInfo*) → *None*

Ensure there is a connection between this host and the peer with given *peer_info.peer_id*. *connect* will absorb the addresses in *peer_info* into its internal peerstore. If there is not an active connection, *connect* will issue a dial, and block until a connection is opened, or an error is returned.

Parameters

peer_info (*peer.peerinfo.PeerInfo*) – *peer_info* of the peer we want to connect to

abstract async disconnect(*peer_id*: *ID*) → *None*

abstract get_addrs() → *List*[*Multiaddr*]

Returns

all the multiaddr addresses this host is listening to

abstract get_id() → *ID*

Returns

peer_id of host

abstract `get_mux()` → *Any*

Returns

mux instance of host

abstract `get_network()` → *INetworkService*

Returns

network instance of host

abstract `get_private_key()` → *PrivateKey*

Returns

the private key belonging to the peer

abstract `get_public_key()` → *PublicKey*

Returns

the public key belonging to the peer

abstract `async new_stream(peer_id: ID, protocol_ids: Sequence[TProtocol])` → *INetStream*

Parameters

- **peer_id** – peer_id that host is connecting
- **protocol_ids** – available protocol ids to use for stream

Returns

stream: new stream created

abstract `run(listen_addrs: Sequence[Multiaddr])` → *AsyncContextManager[None]*

Run the host instance and listen to `listen_addrs`.

Parameters

listen_addrs – a sequence of multiaddrs that we want to listen to

abstract `set_stream_handler(protocol_id: TProtocol, stream_handler: Callable[[INetStream], Awaitable[None]])` → *None*

Set stream handler for host.

Parameters

- **protocol_id** – protocol id used on stream
- **stream_handler** – a stream handler function

libp2p.host.ping module

async `libp2p.host.ping.handle_ping(stream: INetStream)` → *None*

Respond to incoming ping requests until one side errors or closes the `stream`.

libp2p.host.routed_host module

class libp2p.host.routed_host.**RoutedHost**(*network*: [INetworkService](#), *router*: [IPeerRouting](#))

Bases: [BasicHost](#)

async connect(*peer_info*: [PeerInfo](#)) → [None](#)

Ensure there is a connection between this host and the peer with given *peer_info.peer_id*. See ([basic_host](#)).connect for more information.

RoutedHost's Connect differs in that if the host has no addresses for a given peer, it will use its routing system to try to find some.

Parameters

peer_info ([peer.peerinfo.PeerInfo](#)) – peer_info of the peer we want to connect to

Module contents

libp2p.identity package

Subpackages

libp2p.identity.identify package

Subpackages

libp2p.identity.identify.pb package

Submodules

libp2p.identity.identify.pb.identify_pb2 module

Generated protocol buffer code.

class libp2p.identity.identify.pb.identify_pb2.**Identify**

Bases: [Message](#), [Message](#)

DESCRIPTOR = [<google._upb._message.Descriptor object>](#)

Module contents

Submodules

libp2p.identity.identify.protocol module

libp2p.identity.identify.protocol.**identify_handler_for**(*host*: [IHost](#)) → [Callable](#)[[INetStream](#)],
[Awaitable](#)[[None](#)]]

Module contents

Module contents

libp2p.io package

Submodules

libp2p.io.abc module

```
class libp2p.io.abc.Closer
    Bases: ABC
    abstract async close() → None

class libp2p.io.abc.EncryptedMsgReadWriter
    Bases: MsgReadWriteCloser, Encrypter
    Read/write message with encryption/decryption.

class libp2p.io.abc.Encrypter
    Bases: ABC
    abstract decrypt(data: bytes) → bytes
    abstract encrypt(data: bytes) → bytes

class libp2p.io.abc.MsgReadWriteCloser
    Bases: MsgReader, MsgWriter, Closer

class libp2p.io.abc.MsgReader
    Bases: ABC
    abstract async read_msg() → bytes

class libp2p.io.abc.MsgWriter
    Bases: ABC
    abstract async write_msg(msg: bytes) → None

class libp2p.io.abc.ReadCloser
    Bases: Reader, Closer

class libp2p.io.abc.ReadWriteCloser
    Bases: Reader, Writer, Closer

class libp2p.io.abc.ReadWriter
    Bases: Reader, Writer

class libp2p.io.abc.Reader
    Bases: ABC
    abstract async read(n: int | None = None) → bytes

class libp2p.io.abc.WriteCloser
    Bases: Writer, Closer
```

```
class libp2p.io.abc.Writer
```

```
    Bases: ABC
```

```
    abstract async write(data: bytes) → None
```

libp2p.io.exceptions module

```
exception libp2p.io.exceptions.DecryptionFailedException
```

```
    Bases: MsgioException
```

```
exception libp2p.io.exceptions.IOException
```

```
    Bases: BaseLibp2pError
```

```
exception libp2p.io.exceptions.IncompleteReadError
```

```
    Bases: IOException
```

```
    Fewer bytes were read than requested.
```

```
exception libp2p.io.exceptions.MessageTooLarge
```

```
    Bases: MsgioException
```

```
exception libp2p.io.exceptions.MissingLengthException
```

```
    Bases: MsgioException
```

```
exception libp2p.io.exceptions.MissingMessageException
```

```
    Bases: MsgioException
```

```
exception libp2p.io.exceptions.MsgioException
```

```
    Bases: IOException
```

libp2p.io.msgio module

msgio is an implementation of <https://github.com/libp2p/go-msgio>.

from that repo: “a simple package to r/w length-delimited slices.”

NOTE: currently missing the capability to indicate lengths by “varint” method.

```
class libp2p.io.msgio.BaseMsgReadWriter(read_write_closer: ReadWriteCloser)
```

```
    Bases: MsgReadWriteCloser
```

```
    async close() → None
```

```
    abstract encode_msg(msg: bytes) → bytes
```

```
    abstract async next_msg_len() → int
```

```
    async read_msg() → bytes
```

```
    read_write_closer: ReadWriteCloser
```

```
    size_len_bytes: int
```

```
    async write_msg(msg: bytes) → None
```

```
class libp2p.io.msgio.FixedSizeLenMsgReadWriter(read_write_closer: ReadWriteCloser)
```

```
    Bases: BaseMsgReadWriter
```

`encode_msg(msg: bytes) → bytes`

`async next_msg_len() → int`

`size_len_bytes: int`

`class libp2p.io.msgio.VarIntLengthMsgReadWriter(read_write_closer: ReadWriteCloser)`

Bases: `BaseMsgReadWriter`

`encode_msg(msg: bytes) → bytes`

`max_msg_size: int`

`async next_msg_len() → int`

`libp2p.io.msgio.encode_msg_with_length(msg_bytes: bytes, size_len_bytes: int) → bytes`

`async libp2p.io.msgio.read_length(reader: Reader, size_len_bytes: int) → int`

libp2p.io.trio module

`class libp2p.io.trio.TrioTCPStream(stream: SocketStream)`

Bases: `ReadWriteCloser`

`async close() → None`

`async read(n: int | None = None) → bytes`

`read_lock: Lock`

`stream: SocketStream`

`async write(data: bytes) → None`

Raise `RawConnError` if the underlying connection breaks.

`write_lock: Lock`

libp2p.io.utils module

`async libp2p.io.utils.read_exactly(reader: Reader, n: int, retry_count: int = 100) → bytes`

NOTE: relying on exceptions to break out on erroneous conditions, like EOF

Module contents

libp2p.network package

Subpackages

libp2p.network.connection package

Submodules

libp2p.network.connection.exceptions module

exception libp2p.network.connection.exceptions.RawConnError

Bases: *IOException*

libp2p.network.connection.net_connection_interface module

class libp2p.network.connection.net_connection_interface.INetConn

Bases: *Closer*

event_started: *Event*

abstract get_streams() → *Tuple[INetStream, ...]*

muxed_conn: *IMuxedConn*

abstract async new_stream() → *INetStream*

libp2p.network.connection.raw_connection module

class libp2p.network.connection.raw_connection.RawConnection(*stream: ReadWriteCloser, initiator: bool*)

Bases: *IRawConnection*

async close() → *None*

is_initiator: *bool*

async read(*n: int | None = None*) → *bytes*

Read up to *n* bytes from the underlying stream. This call is delegated directly to the underlying self.reader.

Raise *RawConnError* if the underlying connection breaks

stream: *ReadWriteCloser*

async write(*data: bytes*) → *None*

Raise *RawConnError* if the underlying connection breaks.

libp2p.network.connection.raw_connection_interface module

class libp2p.network.connection.raw_connection_interface.IRawConnection

Bases: *ReadWriteCloser*

A Raw Connection provides a Reader and a Writer.

is_initiator: *bool*

libp2p.network.connection.swarm_connection module

```
class libp2p.network.connection.swarm_connection.SwarmConn(muxed_conn: IMuxedConn, swarm: Swarm)
```

Bases: *INetConn*

async `close()` → *None*

event_closed: *Event*

get_streams() → *Tuple[NetStream, ...]*

property is_closed: *bool*

muxed_conn: *IMuxedConn*

async `new_stream()` → *NetStream*

remove_stream(*stream: NetStream*) → *None*

async `start()` → *None*

streams: *Set[NetStream]*

swarm: *Swarm*

Module contents

libp2p.network.stream package

Submodules

libp2p.network.stream.exceptions module

```
exception libp2p.network.stream.exceptions.StreamClosed
```

Bases: *StreamError*

```
exception libp2p.network.stream.exceptions.StreamEOF
```

Bases: *StreamError, EOFError*

```
exception libp2p.network.stream.exceptions.StreamError
```

Bases: *IOException*

```
exception libp2p.network.stream.exceptions.StreamReset
```

Bases: *StreamError*

libp2p.network.stream.net_stream module

class libp2p.network.stream.net_stream.**NetStream**(*muxed_stream*: [IMuxedStream](#))

Bases: [INetStream](#)

async **close**() → [None](#)

Close stream.

get_protocol() → TProtocol

Returns

protocol id that stream runs on

muxed_stream: [IMuxedStream](#)

protocol_id: TProtocol | [None](#)

async **read**(*n*: [int](#) | [None](#) = [None](#)) → [bytes](#)

Read from stream.

Parameters

n – number of bytes to read

Returns

bytes of input

async **reset**() → [None](#)

Close both ends of the stream.

set_protocol(*protocol_id*: TProtocol) → [None](#)

Parameters

protocol_id – protocol id that stream runs on

async **write**(*data*: [bytes](#)) → [None](#)

Write to stream.

Returns

number of bytes written

libp2p.network.stream.net_stream_interface module

class libp2p.network.stream.net_stream_interface.**INetStream**

Bases: [ReadWriteCloser](#)

abstract **get_protocol**() → TProtocol

Returns

protocol id that stream runs on

muxed_conn: [IMuxedConn](#)

abstract **async** **reset**() → [None](#)

Close both ends of the stream.

abstract **set_protocol**(*protocol_id*: TProtocol) → [None](#)

Parameters

protocol_id – protocol id that stream runs on

Module contents

Submodules

libp2p.network.exceptions module

exception libp2p.network.exceptions.SwarmException

Bases: *BaseLibp2pError*

libp2p.network.network_interface module

class libp2p.network.network_interface.INetwork

Bases: *ABC*

abstract async **close**() → *None*

abstract async **close_peer**(*peer_id: ID*) → *None*

connections: *Dict[ID, INetConn]*

abstract async **dial_peer**(*peer_id: ID*) → *INetConn*

dial_peer try to create a connection to peer_id.

Parameters

peer_id – peer if we want to dial

Raises

SwarmException – raised when an error occurs

Returns

muxed connection

abstract **get_peer_id**() → *ID*

Returns

the peer id

abstract async **listen**(**multiaddrs: Sequence[Multiaddr]*) → *bool*

Parameters

multiaddrs – one or many multiaddrs to start listening on

Returns

True if at least one success

listeners: *Dict[str, IListener]*

abstract async **new_stream**(*peer_id: ID*) → *INetStream*

Parameters

- **peer_id** – peer_id of destination
- **protocol_ids** – available protocol ids to use for stream

Returns

net stream instance

peerstore: [IPeerStore](#)

abstract register_notifier(*notifee*: [INotifee](#)) → [None](#)

Parameters

notifee – object implementing Notifee interface

Returns

true if notifee registered successfully, false otherwise

abstract set_stream_handler(*stream_handler*: [Callable](#)[[[INetStream](#)], [Awaitable](#)[[None](#)]]) → [None](#)

Set the stream handler for all incoming streams.

class libp2p.network.network_interface.INetworkService

Bases: [INetwork](#), [ServiceAPI](#)

libp2p.network.notifee_interface module

class libp2p.network.notifee_interface.INotifee

Bases: [ABC](#)

abstract async closed_stream(*network*: [INetwork](#), *stream*: [INetStream](#)) → [None](#)

Parameters

- **network** – network the stream was closed on
- **stream** – stream that was closed

abstract async connected(*network*: [INetwork](#), *conn*: [INetConn](#)) → [None](#)

Parameters

- **network** – network the connection was opened on
- **conn** – connection that was opened

abstract async disconnected(*network*: [INetwork](#), *conn*: [INetConn](#)) → [None](#)

Parameters

- **network** – network the connection was closed on
- **conn** – connection that was closed

abstract async listen(*network*: [INetwork](#), *multiaddr*: [Multiaddr](#)) → [None](#)

Parameters

- **network** – network the listener is listening on
- **multiaddr** – multiaddress listener is listening on

abstract async listen_close(*network*: [INetwork](#), *multiaddr*: [Multiaddr](#)) → [None](#)

Parameters

- **network** – network the connection was opened on
- **multiaddr** – multiaddress listener is no longer listening on

abstract `async opened_stream(network: INetwork, stream: INetStream) → None`

Parameters

- **network** – network the stream was opened on
- **stream** – stream that was opened

libp2p.network.swarm module

class `libp2p.network.swarm.Swarm(peer_id: ID, peerstore: IPeerStore, upgrader: TransportUpgrader, transport: ITransport)`

Bases: `Service`, [INetworkService](#)

async `add_conn(muxed_conn: IMuxedConn) → SwarmConn`

Add a *IMuxedConn* to *Swarm* as a *SwarmConn*, notify “connected”, and start to monitor the connection for its new streams and disconnection.

async `close() → None`

async `close_peer(peer_id: ID) → None`

common_stream_handler: `Callable[[INetStream], Awaitable[None]]`

connections: `Dict[ID, INetConn]`

async `dial_addr(addr: Multiaddr, peer_id: ID) → INetConn`

Try to create a connection to *peer_id* with *addr*.

Parameters

- **addr** – the address we want to connect with
- **peer_id** – the peer we want to connect to

Raises

[SwarmException](#) – raised when an error occurs

Returns

network connection

async `dial_peer(peer_id: ID) → INetConn`

Try to create a connection to *peer_id*.

Parameters

peer_id – peer if we want to dial

Raises

[SwarmException](#) – raised when an error occurs

Returns

muxed connection

event_listener_nursery_created: `Event`

get_peer_id() → [ID](#)

Returns

the peer id

async listen(*multiaddrs: *Multiaddr*) → bool

Parameters

multiaddrs – one or many multiaddrs to start listening on

Returns

true if at least one success

For each multiaddr

- Check if a listener for multiaddr exists already
- If listener already exists, continue
- Otherwise:
 - Capture multiaddr in conn handler
 - Have conn handler delegate to stream handler
 - Call listener listen with the multiaddr
 - Map multiaddr to listener

listener_nursery: *Nursery* | *None*

listeners: *Dict*[*str*, *IListener*]

async new_stream(peer_id: *ID*) → *INetStream*

Parameters

peer_id – peer_id of destination

Raises

SwarmException – raised when an error occurs

Returns

net stream instance

notifees: *List*[*INotiffee*]

async notify_closed_stream(stream: *INetStream*) → *None*

async notify_connected(conn: *INetConn*) → *None*

async notify_disconnected(conn: *INetConn*) → *None*

async notify_listen(multiaddr: *Multiaddr*) → *None*

async notify_listen_close(multiaddr: *Multiaddr*) → *None*

async notify_opened_stream(stream: *INetStream*) → *None*

peerstore: *IPeerStore*

register_notiffee(notiffee: *INotiffee*) → *None*

Parameters

notiffee – object implementing Notiffee interface

Returns

true if notiffee registered successfully, false otherwise

remove_conn(*swarm_conn*: [SwarmConn](#)) → [None](#)

Simply remove the connection from Swarm's records, without closing the connection.

async run() → [None](#)

Primary entry point for all service logic.

Note: This method should **not** be directly invoked by user code.

Services may be run using the following approaches.

self_id: [ID](#)

set_stream_handler(*stream_handler*: [Callable](#)[[[INetStream](#)], [Awaitable](#)[[None](#)]]) → [None](#)

Set the stream handler for all incoming streams.

transport: [ITransport](#)

upgrader: [TransportUpgrader](#)

`libp2p.network.swarm.create_default_stream_handler`(*network*: [INetworkService](#)) →
[Callable](#)[[[INetStream](#)], [Awaitable](#)[[None](#)]]

Module contents

libp2p.peer package

Submodules

libp2p.peer.addrbook_interface module

class `libp2p.peer.addrbook_interface.IAddrBook`

Bases: [ABC](#)

abstract add_addr(*peer_id*: [ID](#), *addr*: [Multiaddr](#), *ttl*: [int](#)) → [None](#)

Calls `add_addrs(peer_id, [addr], ttl)`

Parameters

- **peer_id** – the peer to add address for
- **addr** – multiaddress of the peer
- **ttl** – time-to-live for the address (after this time, address is no longer valid)

abstract add_addrs(*peer_id*: [ID](#), *addrs*: [Sequence](#)[[Multiaddr](#)], *ttl*: [int](#)) → [None](#)

Adds addresses for a given peer all with the same time-to-live. If one of the addresses already exists for the peer and has a longer TTL, no operation should take place. If one of the addresses exists with a shorter TTL, extend the TTL to equal param ttl.

Parameters

- **peer_id** – the peer to add address for
- **addr** – multiaddresses of the peer
- **ttl** – time-to-live for the address (after this time, address is no longer valid)

```
abstract addrs(peer_id: ID) → List[Multiaddr]
```

Parameters
peer_id – peer to get addresses of

Returns
all known (and valid) addresses for the given peer

```
abstract clear_addrs(peer_id: ID) → None
```

Removes all previously stored addresses.

Parameters
peer_id – peer to remove addresses of

```
abstract peers_with_addrs() → List[ID]
```

Returns
all of the peer IDs stored with addresses

libp2p.peer.id module

```
class libp2p.peer.id.ID(peer_id_bytes: bytes)
    Bases: object
    classmethod from_base58(b58_encoded_peer_id_str: str) → ID
    classmethod from_pubkey(key: PublicKey) → ID
    pretty() → str
    to_base58() → str
    to_bytes() → bytes
    to_string() → str
    property xor_id: int
```

```
class libp2p.peer.id.IdentityHash
    Bases: object
    digest() → bytes
    update(input: bytes) → None
```

```
libp2p.peer.id.sha256_digest(data: str | bytes) → bytes
```

libp2p.peer.peerdata module

```
class libp2p.peer.peerdata.PeerData
    Bases: IPeerData
    add_addrs(addrs: Sequence[Multiaddr]) → None
```

Parameters
addrs – multiaddresses to add

add_privkey(privkey: *PrivateKey*) → *None*

Parameters

privkey

add_protocols(protocols: *Sequence[str]*) → *None*

Parameters

protocols – protocols to add

add_pubkey(pubkey: *PublicKey*) → *None*

Parameters

pubkey

addrs: *List[Multiaddr]*

clear_addrs() → *None*

Clear all addresses.

get_addrs() → *List[Multiaddr]*

Returns

all multiaddresses

get_metadata(key: *str*) → *Any*

Parameters

key – key in KV pair

Returns

val for key

Raises

PeerDataError – key not found

get_privkey() → *PrivateKey*

Returns

private key of the peer

Raises

PeerDataError – if private key not found

get_protocols() → *List[str]*

Returns

all protocols associated with given peer

get_pubkey() → *PublicKey*

Returns

public key of the peer

Raises

PeerDataError – if public key not found

metadata: *Dict[Any, Any]*

privkey: *PrivateKey*

protocols: *List[str]*

pubkey: [PublicKey](#)

put_metadata(key: *str*, val: *Any*) → *None*

Parameters

- **key** – key in KV pair
- **val** – val to associate with key

set_protocols(protocols: *Sequence[str]*) → *None*

Parameters

protocols – protocols to set

exception libp2p.peer.peerdata.**PeerDataError**

Bases: [KeyError](#)

Raised when a key is not found in peer metadata.

libp2p.peer.peerdata_interface module

class libp2p.peer.peerdata_interface.**IPeerData**

Bases: [ABC](#)

abstract add_addrs(addrs: *Sequence[Multiaddr]*) → *None*

Parameters

addrs – multiaddresses to add

abstract add_privkey(privkey: [PrivateKey](#)) → *None*

Parameters

privkey

abstract add_protocols(protocols: *Sequence[str]*) → *None*

Parameters

protocols – protocols to add

abstract add_pubkey(pubkey: [PublicKey](#)) → *None*

Parameters

pubkey

abstract clear_addrs() → *None*

Clear all addresses.

abstract get_addrs() → *List[Multiaddr]*

Returns

all multiaddresses

abstract get_metadata(key: *str*) → *IPeerMetadata*

Parameters

key – key in KV pair

Returns

val for key

Raises*PeerDataError* – key not found**abstract** `get_privkey()` → *PrivateKey***Returns**

private key of the peer

Raises*PeerDataError* – if private key not found**abstract** `get_protocols()` → *List[str]***Returns**

all protocols associated with given peer

abstract `get_pubkey()` → *PublicKey***Returns**

public key of the peer

Raises*PeerDataError* – if public key not found**abstract** `put_metadata(key: str, val: Any)` → *None***Parameters**

- **key** – key in KV pair
- **val** – val to associate with key

abstract `set_protocols(protocols: Sequence[str])` → *None***Parameters****protocols** – protocols to set**libp2p.peer.peerinfo module****exception** `libp2p.peer.peerinfo.InvalidAddrError`Bases: *ValueError***class** `libp2p.peer.peerinfo.PeerInfo(peer_id: ID, addrs: Sequence[Multiaddr])`Bases: *object***addrs:** *List[Multiaddr]***peer_id:** *ID*`libp2p.peer.peerinfo.info_from_p2p_addr(addr: Multiaddr)` → *PeerInfo*

libp2p.peer.peermetadata_interface module

class libp2p.peer.peermetadata_interface.**IPeerMetadata**

Bases: [ABC](#)

abstract **get**(*peer_id*: [ID](#), *key*: *str*) → *Any*

Parameters

- **peer_id** – peer ID to lookup key for
- **key** – key to look up

Returns

value at key for given peer

Raises

[Exception](#) – peer ID not found

abstract **put**(*peer_id*: [ID](#), *key*: *str*, *val*: *Any*) → *None*

Parameters

- **peer_id** – peer ID to lookup key for
- **key** – key to associate with peer
- **val** – value to associated with key

Raises

[Exception](#) – unsuccessful put

libp2p.peer.peerstore module

class libp2p.peer.peerstore.**PeerStore**

Bases: [IPeerStore](#)

add_addr(*peer_id*: [ID](#), *addr*: *Multiaddr*, *ttl*: *int*) → *None*

Parameters

- **peer_id** – peer ID to add address for
- **addr**
- **ttl** – time-to-live for the this record

add_addrs(*peer_id*: [ID](#), *addrs*: *Sequence*[*Multiaddr*], *ttl*: *int*) → *None*

Parameters

- **peer_id** – peer ID to add address for
- **addrs**
- **ttl** – time-to-live for the this record

add_key_pair(*peer_id*: [ID](#), *key_pair*: [KeyPair](#)) → *None*

Parameters

- **peer_id** – peer ID to add private key for
- **key_pair**

add_privkey(*peer_id*: ID, *privkey*: PrivateKey) → None

Parameters

- **peer_id** – peer ID to add private key for
- **privkey**

Raises

PeerStoreError – if peer ID or peer privkey not found

add_protocols(*peer_id*: ID, *protocols*: Sequence[str]) → None

Parameters

- **peer_id** – peer ID to add protocols for
- **protocols** – protocols to add

add_pubkey(*peer_id*: ID, *pubkey*: PublicKey) → None

Parameters

- **peer_id** – peer ID to add public key for
- **pubkey**

Raises

PeerStoreError – if peer ID and pubkey does not match

addrs(*peer_id*: ID) → List[Multiaddr]

Parameters

peer_id – peer ID to get addrs for

Returns

list of addrs

Raises

PeerStoreError – if peer ID not found

clear_addrs(*peer_id*: ID) → None

Parameters

peer_id – peer ID to clear addrs for

get(*peer_id*: ID, *key*: str) → Any

Parameters

- **peer_id** – peer ID to get peer data for
- **key** – the key to search value for

Returns

value corresponding to the key

Raises

PeerStoreError – if peer ID or value not found

get_protocols(*peer_id*: ID) → List[str]

Parameters

peer_id – peer ID to get protocols for

Returns

protocols (as list of strings)

Raises

PeerStoreError – if peer ID not found

peer_data_map: `Dict[ID, PeerData]`

peer_ids() → `List[ID]`

Returns

all of the peer IDs stored in peer store

peer_info(*peer_id*: ID) → *PeerInfo*

Parameters

peer_id – peer ID to get info for

Returns

peer info object

peers_with_addrs() → `List[ID]`

Returns

all of the peer IDs which has addrs stored in peer store

privkey(*peer_id*: ID) → *PrivateKey*

Parameters

peer_id – peer ID to get private key for

Returns

private key of the peer

Raises

PeerStoreError – if peer ID or peer privkey not found

pubkey(*peer_id*: ID) → *PublicKey*

Parameters

peer_id – peer ID to get public key for

Returns

public key of the peer

Raises

PeerStoreError – if peer ID or peer pubkey not found

put(*peer_id*: ID, *key*: str, *val*: Any) → None

Parameters

- **peer_id** – peer ID to put peer data for
- **key**
- **value**

set_protocols(*peer_id*: ID, *protocols*: Sequence[str]) → None

Parameters

- **peer_id** – peer ID to set protocols for
- **protocols** – protocols to set

exception libp2p.peer.peerstore.**PeerStoreError**

Bases: [KeyError](#)

Raised when peer ID is not found in peer store.

libp2p.peer.peerstore_interface module

class libp2p.peer.peerstore_interface.**IPeerStore**

Bases: [IAddrBook](#), [IPeerMetadata](#)

abstract **add_addr**(*peer_id*: [ID](#), *addr*: [Multiaddr](#), *ttl*: [int](#)) → [None](#)

Parameters

- **peer_id** – peer ID to add address for
- **addr**
- **ttl** – time-to-live for the this record

abstract **add_addrs**(*peer_id*: [ID](#), *addrs*: [Sequence](#)[[Multiaddr](#)], *ttl*: [int](#)) → [None](#)

Parameters

- **peer_id** – peer ID to add address for
- **addrs**
- **ttl** – time-to-live for the this record

abstract **add_key_pair**(*peer_id*: [ID](#), *key_pair*: [KeyPair](#)) → [None](#)

Parameters

- **peer_id** – peer ID to add private key for
- **key_pair**

Raises

[PeerStoreError](#) – if peer ID already has pubkey or privkey set

abstract **add_privkey**(*peer_id*: [ID](#), *privkey*: [PrivateKey](#)) → [None](#)

Parameters

- **peer_id** – peer ID to add private key for
- **privkey**

Raises

[PeerStoreError](#) – if peer ID already has privkey set

abstract **add_protocols**(*peer_id*: [ID](#), *protocols*: [Sequence](#)[[str](#)]) → [None](#)

Parameters

- **peer_id** – peer ID to add protocols for
- **protocols** – protocols to add

abstract **add_pubkey**(*peer_id*: [ID](#), *pubkey*: [PublicKey](#)) → [None](#)

Parameters

- **peer_id** – peer ID to add public key for

- **pubkey**

Raises

PeerStoreError – if peer ID already has pubkey set

abstract **addrs**(*peer_id*: ID) → List[Multiaddr]

Parameters

peer_id – peer ID to get addrs for

Returns

list of addrs

abstract **clear_addrs**(*peer_id*: ID) → None

Parameters

peer_id – peer ID to clear addrs for

abstract **get**(*peer_id*: ID, *key*: str) → Any

Parameters

- **peer_id** – peer ID to get peer data for
- **key** – the key to search value for

Returns

value corresponding to the key

Raises

PeerStoreError – if peer ID or value not found

abstract **get_protocols**(*peer_id*: ID) → List[str]

Parameters

peer_id – peer ID to get protocols for

Returns

protocols (as list of strings)

Raises

PeerStoreError – if peer ID not found

abstract **peer_ids**() → List[ID]

Returns

all of the peer IDs stored in peer store

abstract **peer_info**(*peer_id*: ID) → *PeerInfo*

Parameters

peer_id – peer ID to get info for

Returns

peer info object

abstract **peers_with_addrs**() → List[ID]

Returns

all of the peer IDs which has addrs stored in peer store

abstract `privkey(peer_id: ID) → PrivateKey`

Parameters

peer_id – peer ID to get private key for

Returns

private key of the peer

Raises

PeerStoreError – if peer ID not found

abstract `pubkey(peer_id: ID) → PublicKey`

Parameters

peer_id – peer ID to get public key for

Returns

public key of the peer

Raises

PeerStoreError – if peer ID not found

abstract `put(peer_id: ID, key: str, val: Any) → None`

Parameters

- **peer_id** – peer ID to put peer data for
- **key**
- **value**

abstract `set_protocols(peer_id: ID, protocols: Sequence[str]) → None`

Parameters

- **peer_id** – peer ID to set protocols for
- **protocols** – protocols to set

Module contents

libp2p.protocol_muxer package

Submodules

libp2p.protocol_muxer.exceptions module

exception `libp2p.protocol_muxer.exceptions.MultiselectClientError`

Bases: *BaseLibp2pError*

Raised when an error occurs in protocol selection process.

exception `libp2p.protocol_muxer.exceptions.MultiselectCommunicatorError`

Bases: *BaseLibp2pError*

Raised when an error occurs during read/write via communicator.

exception libp2p.protocol_muxer.exceptions.**MultiselectError**

Bases: *BaseLibp2pError*

Raised when an error occurs in multiselect process.

libp2p.protocol_muxer.multiselect module

class libp2p.protocol_muxer.multiselect.**Multiselect**(*default_handlers: Dict[TProtocol, Callable[[INetStream], Awaitable[None]]] = None*)

Bases: *IMultiselectMuxer*

Multiselect module that is responsible for responding to a multiselect client and deciding on a specific protocol and handler pair to use for communication.

add_handler(*protocol: TProtocol, handler: Callable[[INetStream], Awaitable[None]]*) → None

Store the handler with the given protocol.

Parameters

- **protocol** – protocol name
- **handler** – handler function

handlers: Dict[TProtocol, Callable[[INetStream], Awaitable[None]]]

async handshake(*communicator: IMultiselectCommunicator*) → None

Perform handshake to agree on multiselect protocol.

Parameters

communicator – communicator to use

Raises

MultiselectError – raised when handshake failed

async negotiate(*communicator: IMultiselectCommunicator*) → Tuple[TProtocol, Callable[[INetStream], Awaitable[None]]]

Negotiate performs protocol selection.

Parameters

stream – stream to negotiate on

Returns

selected protocol name, handler function

Raises

MultiselectError – raised when negotiation failed

libp2p.protocol_muxer.multiselect.**is_valid_handshake**(*handshake_contents: str*) → bool

Determine if handshake is valid and should be confirmed.

Parameters

handshake_contents – contents of handshake message

Returns

true if handshake is complete, false otherwise

libp2p.protocol_muxer.multiselect_client module**class** libp2p.protocol_muxer.multiselect_client.**MultiselectClient**Bases: *IMultiselectClient*

Client for communicating with receiver's multiselect module in order to select a protocol id to communicate over.

async **handshake**(communicator: *IMultiselectCommunicator*) → *None*

Ensure that the client and multiselect are both using the same multiselect protocol.

Parameters

stream – stream to communicate with multiselect over

Raises

MultiselectClientError – raised when handshake failed

async **select_one_of**(protocols: *Sequence[TProtocol]*, communicator: *IMultiselectCommunicator*) → *TProtocol*

For each protocol, send message to multiselect selecting protocol and fail if multiselect does not return same protocol. Returns first protocol that multiselect agrees on (i.e. that multiselect selects)

Parameters

- **protocol** – protocol to select
- **stream** – stream to communicate with multiselect over

Returns

selected protocol

Raises

MultiselectClientError – raised when protocol negotiation failed

async **try_select**(communicator: *IMultiselectCommunicator*, protocol: *TProtocol*) → *TProtocol*

Try to select the given protocol or raise exception if fails.

Parameters

- **communicator** – communicator to use to communicate with counterparty
- **protocol** – protocol to select

Raises

MultiselectClientError – raised when protocol negotiation failed

Returns

selected protocol

libp2p.protocol_muxer.multiselect_client.is_valid_handshake(handshake_contents: *str*) → *bool*

Determine if handshake is valid and should be confirmed.

Parameters

handshake_contents – contents of handshake message

Returns

true if handshake is complete, false otherwise

libp2p.protocol_muxer.multiselect_client_interface module

class libp2p.protocol_muxer.multiselect_client_interface.**IMultiselectClient**

Bases: [ABC](#)

Client for communicating with receiver's multiselect module in order to select a protocol id to communicate over.

abstract async handshake(communicator: [IMultiselectCommunicator](#)) → [None](#)

Ensure that the client and multiselect are both using the same multiselect protocol.

Parameters

stream – stream to communicate with multiselect over

Raises

[Exception](#) – multiselect protocol ID mismatch

abstract async select_one_of(protocols: [Sequence](#)[[TProtocol](#)], communicator: [IMultiselectCommunicator](#)) → [TProtocol](#)

For each protocol, send message to multiselect selecting protocol and fail if multiselect does not return same protocol. Returns first protocol that multiselect agrees on (i.e. that multiselect selects)

Parameters

- **protocol** – protocol to select
- **stream** – stream to communicate with multiselect over

Returns

selected protocol

abstract async try_select(communicator: [IMultiselectCommunicator](#), protocol: [TProtocol](#)) → [TProtocol](#)

Try to select the given protocol or raise exception if fails.

Parameters

- **communicator** – communicator to use to communicate with counterparty
- **protocol** – protocol to select

Raises

[Exception](#) – error in protocol selection

Returns

selected protocol

libp2p.protocol_muxer.multiselect_communicator module

class libp2p.protocol_muxer.multiselect_communicator.**MultiselectCommunicator**(read_writer: [ReadWriteCloser](#))

Bases: [IMultiselectCommunicator](#)

async read() → [str](#)

Raises

[MultiselectCommunicatorError](#) – raised when failed to read from underlying reader

read_writer: *ReadWriteCloser*

async write(*msg_str: str*) → None

Raises

MultiselectCommunicatorError – raised when failed to write to underlying reader

libp2p.protocol_muxer.multiselect_communicator_interface module

class libp2p.protocol_muxer.multiselect_communicator_interface.**IMultiselectCommunicator**

Bases: *ABC*

Communicator helper class that ensures both the client and multistream module will follow the same multistream protocol, which is necessary for them to work.

abstract async read() → *str*

Reads message from stream until EOF.

abstract async write(*msg_str: str*) → None

Write message to stream.

Parameters

msg_str – message to write

libp2p.protocol_muxer.multiselect_muxer_interface module

class libp2p.protocol_muxer.multiselect_muxer_interface.**IMultiselectMuxer**

Bases: *ABC*

Multiselect module that is responsible for responding to a multiselect client and deciding on a specific protocol and handler pair to use for communication.

abstract add_handler(*protocol: TProtocol, handler: Callable[[INetStream], Awaitable[None]]*) → None

Store the handler with the given protocol.

Parameters

- **protocol** – protocol name
- **handler** – handler function

get_protocols() → *Tuple*[*TProtocol*, ...]

handlers: *Dict*[*TProtocol*, *Callable*[[*INetStream*], *Awaitable*[*None*]]]

abstract async negotiate(*communicator: IMultiselectCommunicator*) → *Tuple*[*TProtocol*, *Callable*[[*INetStream*], *Awaitable*[*None*]]]

Negotiate performs protocol selection.

Parameters

stream – stream to negotiate on

Returns

selected protocol name, handler function

Raises

Exception – negotiation failed exception

Module contents

libp2p.pubsub package

Subpackages

libp2p.pubsub.pb package

Submodules

libp2p.pubsub.pb.rpc_pb2 module

Generated protocol buffer code.

```
class libp2p.pubsub.pb.rpc_pb2.ControlGraft
    Bases: Message, Message
    DESCRIPTOR = <google._upb._message.Descriptor object>

class libp2p.pubsub.pb.rpc_pb2.ControlIHave
    Bases: Message, Message
    DESCRIPTOR = <google._upb._message.Descriptor object>

class libp2p.pubsub.pb.rpc_pb2.ControlIWant
    Bases: Message, Message
    DESCRIPTOR = <google._upb._message.Descriptor object>

class libp2p.pubsub.pb.rpc_pb2.ControlMessage
    Bases: Message, Message
    DESCRIPTOR = <google._upb._message.Descriptor object>

class libp2p.pubsub.pb.rpc_pb2.ControlPrune
    Bases: Message, Message
    DESCRIPTOR = <google._upb._message.Descriptor object>

class libp2p.pubsub.pb.rpc_pb2.Message
    Bases: Message, Message
    DESCRIPTOR = <google._upb._message.Descriptor object>

class libp2p.pubsub.pb.rpc_pb2.RPC
    Bases: Message, Message
    DESCRIPTOR = <google._upb._message.Descriptor object>

    class SubOpts
        Bases: Message, Message
        DESCRIPTOR = <google._upb._message.Descriptor object>

class libp2p.pubsub.pb.rpc_pb2.TopicDescriptor
    Bases: Message, Message
```

```
class AuthOpts
    Bases: Message, Message
    DESCRIPTOR = <google._upb._message.Descriptor object>
DESCRIPTOR = <google._upb._message.Descriptor object>

class EncOpts
    Bases: Message, Message
    DESCRIPTOR = <google._upb._message.Descriptor object>
```

Module contents

Submodules

libp2p.pubsub.abc module

```
class libp2p.pubsub.abc.IPubsub
    Bases: ServiceAPI
    abstract property my_id: ID
    abstract property protocols: Tuple[TProtocol, ...]
    abstract async publish(topic_id: str, data: bytes) → None
    abstract remove_topic_validator(topic: str) → None
    abstract set_topic_validator(topic: str, validator: Callable[[ID, Message], bool] | Callable[[ID, Message], Awaitable[bool]], is_async_validator: bool) → None
    abstract async subscribe(topic_id: str) → ISubscriptionAPI
    abstract property topic_ids: KeysView[str]
    abstract async unsubscribe(topic_id: str) → None
    abstract async wait_until_ready() → None

class libp2p.pubsub.abc.IPubsubRouter
    Bases: ABC
    abstract add_peer(peer_id: ID, protocol_id: TProtocol) → None
        Notifies the router that a new peer has been connected.

        Parameters
            peer_id – id of peer to add
    abstract attach(pubsub: Pubsub) → None
        Attach is invoked by the PubSub constructor to attach the router to a freshly initialized PubSub instance.

        Parameters
            pubsub – pubsub instance to attach to
```

abstract `get_protocols()` → `List[TProtocol]`

Returns

the list of protocols supported by the router

abstract `async handle_rpc(rpc: RPC, sender_peer_id: ID)` → `None`

Invoked to process control messages in the RPC envelope. It is invoked after subscriptions and payload messages have been processed TODO: Check if this interface is ok. It's not the exact same as the go code, but the go code is really confusing with the msg origin, they specify *rpc.from* even when the rpc shouldn't have a from :param rpc: rpc message

abstract `async join(topic: str)` → `None`

Join notifies the router that we want to receive and forward messages in a topic. It is invoked after the subscription announcement.

Parameters

topic – topic to join

abstract `async leave(topic: str)` → `None`

Leave notifies the router that we are no longer interested in a topic. It is invoked after the unsubscription announcement.

Parameters

topic – topic to leave

abstract `async publish(msg_forwarder: ID, pubsub_msg: Message)` → `None`

Invoked to forward a new message that has been validated.

Parameters

- **msg_forwarder** – peer_id of message sender
- **pubsub_msg** – pubsub message to forward

abstract `remove_peer(peer_id: ID)` → `None`

Notifies the router that a peer has been disconnected.

Parameters

peer_id – id of peer to remove

class `libp2p.pubsub.abc.ISubscriptionAPI`

Bases: `AsyncContextManager[ISubscriptionAPI]`, `AsyncIterable[Message]`

abstract `async get()` → `Message`

abstract `async unsubscribe()` → `None`

libp2p.pubsub.exceptions module

exception `libp2p.pubsub.exceptions.NoPubsubAttached`

Bases: `PubsubRouterError`

exception `libp2p.pubsub.exceptions.PubsubRouterError`

Bases: `BaseLibp2pError`

libp2p.pubsub.floodsub module

class libp2p.pubsub.floodsub.FloodSub(protocols: *Sequence*[TProtocol])

Bases: *IPubsubRouter*

add_peer(peer_id: ID, protocol_id: TProtocol) → None

Notifies the router that a new peer has been connected.

Parameters

peer_id – id of peer to add

attach(pubsub: Pubsub) → None

Attach is invoked by the PubSub constructor to attach the router to a freshly initialized PubSub instance.

Parameters

pubsub – pubsub instance to attach to

get_protocols() → List[TProtocol]

Returns

the list of protocols supported by the router

async handle_rpc(rpc: RPC, sender_peer_id: ID) → None

Invoked to process control messages in the RPC envelope. It is invoked after subscriptions and payload messages have been processed.

Parameters

rpc – rpc message

async join(topic: str) → None

Join notifies the router that we want to receive and forward messages in a topic. It is invoked after the subscription announcement.

Parameters

topic – topic to join

async leave(topic: str) → None

Leave notifies the router that we are no longer interested in a topic. It is invoked after the unsubscription announcement.

Parameters

topic – topic to leave

protocols: List[TProtocol]

async publish(msg_forwarder: ID, pubsub_msg: Message) → None

Invoked to forward a new message that has been validated. This is where the “flooding” part of floodsub happens.

With flooding, routing is almost trivial: for each incoming message, forward to all known peers in the topic. There is a bit of logic, as the router maintains a timed cache of previous messages, so that seen messages are not further forwarded. It also never forwards a message back to the source or the peer that forwarded the message. :param msg_forwarder: peer ID of the peer who forwards the message to us :param pubsub_msg: pubsub message in protobuf.

pubsub: Pubsub

remove_peer(*peer_id*: ID) → None

Notifies the router that a peer has been disconnected.

Parameters

peer_id – id of peer to remove

libp2p.pubsub.gossipsub module

class libp2p.pubsub.gossipsub.**GossipSub**(*protocols*: Sequence[TProtocol], *degree*: int, *degree_low*: int, *degree_high*: int, *time_to_live*: int, *gossip_window*: int = 3, *gossip_history*: int = 5, *heartbeat_initial_delay*: float = 0.1, *heartbeat_interval*: int = 120)

Bases: *IPubsubRouter*, *Service*

add_peer(*peer_id*: ID, *protocol_id*: TProtocol) → None

Notifies the router that a new peer has been connected.

Parameters

- **peer_id** – id of peer to add
- **protocol_id** – router protocol the peer speaks, e.g., floodsub, gossipsub

attach(*pubsub*: Pubsub) → None

Attach is invoked by the PubSub constructor to attach the router to a freshly initialized PubSub instance.

Parameters

pubsub – pubsub instance to attach to

degree: int

degree_high: int

degree_low: int

async emit_control_message(*control_msg*: ControlMessage, *to_peer*: ID) → None

async emit_graft(*topic*: str, *to_peer*: ID) → None

Emit graft message, sent to to_peer, for topic.

async emit_ihave(*topic*: str, *msg_ids*: Any, *to_peer*: ID) → None

Emit ihave message, sent to to_peer, for topic and msg_ids.

async emit_iwant(*msg_ids*: Any, *to_peer*: ID) → None

Emit iwant message, sent to to_peer, for msg_ids.

async emit_prune(*topic*: str, *to_peer*: ID) → None

Emit graft message, sent to to_peer, for topic.

fanout: Dict[str, Set[ID]]

fanout_heartbeat() → None

get_protocols() → List[TProtocol]

Returns

the list of protocols supported by the router

gossip_heartbeat() → `DefaultDict[ID, Dict[str, List[str]]]`

async handle_graft(*graft_msg*: `ControlGraft`, *sender_peer_id*: `ID`) → `None`

async handle_ihave(*ihave_msg*: `ControlIHave`, *sender_peer_id*: `ID`) → `None`

Checks the seen set and requests unknown messages with an IWANT message.

async handle_iwant(*iwant_msg*: `ControlIWant`, *sender_peer_id*: `ID`) → `None`

Forwards all request messages that are present in mcache to the requesting peer.

async handle_prune(*prune_msg*: `ControlPrune`, *sender_peer_id*: `ID`) → `None`

async handle_rpc(*rpc*: `RPC`, *sender_peer_id*: `ID`) → `None`

Invoked to process control messages in the RPC envelope. It is invoked after subscriptions and payload messages have been processed.

Parameters

- **rpc** – RPC message
- **sender_peer_id** – id of the peer who sent the message

async heartbeat() → `None`

Call individual heartbeats.

Note: the heartbeats are called with awaits because each heartbeat depends on the state changes in the preceding heartbeat

heartbeat_initial_delay: `float`

heartbeat_interval: `int`

async join(*topic*: `str`) → `None`

Join notifies the router that we want to receive and forward messages in a topic. It is invoked after the subscription announcement.

Parameters

topic – topic to join

async leave(*topic*: `str`) → `None`

Leave notifies the router that we are no longer interested in a topic. It is invoked after the unsubscription announcement.

Parameters

topic – topic to leave

mcache: `MessageCache`

mesh: `Dict[str, Set[ID]]`

mesh_heartbeat() → `Tuple[DefaultDict[ID, List[str]], DefaultDict[ID, List[str]]]`

pack_control_msgs(*ihave_msgs*: `List[ControlIHave]`, *graft_msgs*: `List[ControlGraft]`, *prune_msgs*: `List[ControlPrune]`) → `ControlMessage`

peer_protocol: `Dict[ID, TProtocol]`

protocols: `List[TProtocol]`

async publish(*msg_forwarder*: `ID`, *pubsub_msg*: `Message`) → `None`

Invoked to forward a new message that has been validated.

pubsub: *Pubsub*

remove_peer(*peer_id*: ID) → None

Notifies the router that a peer has been disconnected.

Parameters

peer_id – id of peer to remove

async run() → None

Primary entry point for all service logic.

Note: This method should **not** be directly invoked by user code.

Services may be run using the following approaches.

static select_from_minus(*num_to_select*: int, *pool*: Iterable[Any], *minus*: Iterable[Any]) → List[Any]

Select at most num_to_select subset of elements from the set (pool - minus) randomly. :param num_to_select: number of elements to randomly select :param pool: list of items to select from (excluding elements in minus) :param minus: elements to be excluded from selection pool :return: list of selected elements

time_to_live: int

libp2p.pubsub.mcache module

class libp2p.pubsub.mcache.CacheEntry(*mid*: Tuple[bytes, bytes], *topics*: Sequence[str])

Bases: object

mid: Tuple[bytes, bytes]

topics: List[str]

A logical representation of an entry in the mcache's _history_.

class libp2p.pubsub.mcache.MessageCache(*window_size*: int, *history_size*: int)

Bases: object

get(*mid*: Tuple[bytes, bytes]) → Message | None

Get a message from the mcache.

Parameters

mid – (seqno, from_id) of the message to get.

Returns

The rpc message associated with this mid

history: List[List[CacheEntry]]

history_size: int

msgs: Dict[Tuple[bytes, bytes], Message]

put(*msg*: Message) → None

Put a message into the mcache.

Parameters

msg – The rpc message to put in. Should contain seqno and from_id

shift() → *None*

Shift the window over by 1 position, dropping the last element of the history.

window(topic: *str*) → *List[Tuple[bytes, bytes]]*

Get the window for this topic.

Parameters

topic – Topic whose message ids we desire.

Returns

List of mids in the current window.

window_size: *int*

libp2p.pubsub.pubsub module

```
class libp2p.pubsub.pubsub.Pubsub(host: ~libp2p.host.host_interface.IHost, router: IPubsubRouter,
    cache_size: int = None, strict_signing: bool = True,
    msg_id_constructor:
    ~typing.Callable[[-libp2p.pubsub.pb.rpc_pb2.Message], bytes] =
    <function get_peer_and_seqno_msg_id>)
```

Bases: Service, *IPubsub*

async continuously_read_stream(stream: *INetStream*) → *None*

Read from input stream in an infinite loop. Process messages from other nodes.

Parameters

stream – stream to continuously read from

counter: *int*

dead_peer_receive_channel: *trio.MemoryReceiveChannel[*ID*]*

event_handle_dead_peer_queue_started: *Event*

event_handle_peer_queue_started: *Event*

get_hello_packet() → *RPC*

Generate subscription message with all topics we are subscribed to only send hello packet if we have subscribed topics.

get_msg_validators(msg: *Message*) → *Tuple[TopicValidator, ...]*

Get all validators corresponding to the topics in the message.

Parameters

msg – the message published to the topic

async handle_dead_peer_queue() → *None*

Continuously read from dead peer channel and close the stream between that peer and remove peer info from pubsub and pubsub router.

async handle_peer_queue() → *None*

Continuously read from peer queue and each time a new peer is found, open a stream to the peer using a supported pubsub protocol pubsub protocols we support.

handle_subscription(*origin_id*: [ID](#), *sub_message*: [SubOpts](#)) → [None](#)

Handle an incoming subscription message from a peer. Update internal mapping to mark the peer as subscribed or unsubscribed to topics as defined in the subscription message.

Parameters

- **origin_id** – id of the peer who subscribe to the message
- **sub_message** – [RPC.SubOpts](#)

host: [IHost](#)

async message_all_peers(*raw_msg*: [bytes](#)) → [None](#)

Broadcast a message to peers.

Parameters

raw_msg – raw contents of the message to broadcast

property my_id: [ID](#)

notify_subscriptions(*publish_message*: [Message](#)) → [None](#)

Put incoming message from a peer onto my blocking queue.

Parameters

publish_message – [RPC.Message](#) format

peer_receive_channel: [trio.MemoryReceiveChannel](#)[[ID](#)]

peer_topics: [Dict](#)[[str](#), [Set](#)[[ID](#)]]

peers: [Dict](#)[[ID](#), [INetStream](#)]

property protocols: [Tuple](#)[[TProtocol](#), ...]

async publish(*topic_id*: [str](#), *data*: [bytes](#)) → [None](#)

Publish data to a topic.

Parameters

- **topic_id** – topic which we are going to publish the data to
- **data** – data which we are publishing

async push_msg(*msg_forwarder*: [ID](#), *msg*: [Message](#)) → [None](#)

Push a pubsub message to others.

Parameters

- **msg_forwarder** – the peer who forward us the message.
- **msg** – the message we are going to push out.

remove_topic_validator(*topic*: [str](#)) → [None](#)

Remove the validator from the given topic.

Parameters

topic – the topic to remove validator from

router: [IPubsubRouter](#)

async run() → *None*

Primary entry point for all service logic.

Note: This method should **not** be directly invoked by user code.

Services may be run using the following approaches.

seen_messages: *LRU*

set_topic_validator(*topic: str, validator: Callable[[ID, Message], bool] | Callable[[ID, Message], Awaitable[bool]], is_async_validator: bool*) → *None*

Register a validator under the given topic. One topic can only have one validator.

Parameters

- **topic** – the topic to register validator under
- **validator** – the validator used to validate messages published to the topic
- **is_async_validator** – indicate if the validator is an asynchronous validator

sign_key: *PrivateKey*

async stream_handler(*stream: INetStream*) → *None*

Stream handler for pubsub. Gets invoked whenever a new stream is created on one of the supported pubsub protocols.

Parameters

stream – newly created stream

strict_signing: *bool*

async subscribe(*topic_id: str*) → *ISubscriptionAPI*

Subscribe ourselves to a topic.

Parameters

topic_id – topic_id to subscribe to

subscribed_topics_receive: *Dict[str, TrioSubscriptionAPI]*

subscribed_topics_send: *Dict[str, trio.MemorySendChannel[*rpc_pb2.Message*]]*

property topic_ids: *KeysView[str]*

topic_validators: *Dict[str, TopicValidator]*

async unsubscribe(*topic_id: str*) → *None*

Unsubscribe ourselves from a topic.

Parameters

topic_id – topic_id to unsubscribe from

async validate_msg(*msg_forwarder: ID, msg: Message*) → *None*

Validate the received message.

Parameters

- **msg_forwarder** – the peer who forward us the message.
- **msg** – the message.

`async wait_until_ready()` → `None`

`class libp2p.pubsub.pubsub.TopicValidator(validator, is_async)`

Bases: `NamedTuple`

`is_async`: `bool`

Alias for field number 1

`validator`: `Callable[[ID, Message], bool] | Callable[[ID, Message], Awaitable[bool]]`

Alias for field number 0

`libp2p.pubsub.pubsub.get_content_addressed_msg_id(msg: Message)` → `bytes`

`libp2p.pubsub.pubsub.get_peer_and_seqno_msg_id(msg: Message)` → `bytes`

libp2p.pubsub.pubsub_notiffee module

`class libp2p.pubsub.pubsub_notiffee.PubsubNotiffee(initiator_peers_queue:
 trio.MemorySendChannel[ID], dead_peers_queue:
 trio.MemorySendChannel[ID])`

Bases: `INotiffee`

`async closed_stream(network: INetwork, stream: INetStream)` → `None`

Parameters

- **network** – network the stream was closed on
- **stream** – stream that was closed

`async connected(network: INetwork, conn: INetConn)` → `None`

Add `peer_id` to `initiator_peers_queue`, so that this `peer_id` can be used to create a stream and we only want to have one pubsub stream with each peer.

Parameters

- **network** – network the connection was opened on
- **conn** – connection that was opened

`dead_peers_queue`: `trio.MemorySendChannel[ID]`

`async disconnected(network: INetwork, conn: INetConn)` → `None`

Add `peer_id` to `dead_peers_queue`, so that pubsub and its router can remove this `peer_id` and close the stream inbetween.

Parameters

- **network** – network the connection was opened on
- **conn** – connection that was opened

`initiator_peers_queue`: `trio.MemorySendChannel[ID]`

`async listen(network: INetwork, multiaddr: Multiaddr)` → `None`

Parameters

- **network** – network the listener is listening on
- **multiaddr** – multiaddress listener is listening on

async listen_close(*network*: [INetwork](#), *multiaddr*: [Multiaddr](#)) → [None](#)

Parameters

- **network** – network the connection was opened on
- **multiaddr** – multiaddress listener is no longer listening on

async opened_stream(*network*: [INetwork](#), *stream*: [INetStream](#)) → [None](#)

Parameters

- **network** – network the stream was opened on
- **stream** – stream that was opened

libp2p.pubsub.subscription module

class libp2p.pubsub.subscription.**BaseSubscriptionAPI**

Bases: [ISubscriptionAPI](#)

class libp2p.pubsub.subscription.**TrioSubscriptionAPI**(*receive_channel*:
[trio.MemoryReceiveChannel](#)[[rpc_pb2.Message](#)],
unsubscribe_fn: [Callable](#)[[],
[Awaitable](#)[[None](#)]])

Bases: [BaseSubscriptionAPI](#)

async get() → [Message](#)

receive_channel: [trio.MemoryReceiveChannel](#)[[rpc_pb2.Message](#)]

async unsubscribe() → [None](#)

unsubscribe_fn: [Callable](#)[[], [Awaitable](#)[[None](#)]]

libp2p.pubsub.typing module

libp2p.pubsub.validators module

libp2p.pubsub.validators.**signature_validator**(*msg*: [Message](#)) → [bool](#)

Verify the message against the given public key.

Parameters

- **pubkey** – the public key which signs the message.
- **msg** – the message signed.

Module contents

libp2p.routing package

Submodules

libp2p.routing.interfaces module

class libp2p.routing.interfaces.IContentRouting

Bases: [ABC](#)

abstract find_provider_iter(*cid: bytes, count: int*) → [Iterable\[PeerInfo\]](#)

Search for peers who are able to provide a given key returns an iterator of peer.PeerInfo.

abstract provide(*cid: bytes, announce: bool = True*) → [None](#)

Provide adds the given cid to the content routing system.

If announce is True, it also announces it, otherwise it is just kept in the local accounting of which objects are being provided.

class libp2p.routing.interfaces.IPeerRouting

Bases: [ABC](#)

abstract async find_peer(*peer_id: ID*) → [PeerInfo](#)

Find specific Peer FindPeer searches for a peer with given peer_id, returns a peer.PeerInfo with relevant addresses.

Module contents

libp2p.security package

Subpackages

libp2p.security.insecure package

Subpackages

libp2p.security.insecure.pb package

Submodules

libp2p.security.insecure.pb.plaintext_pb2 module

Generated protocol buffer code.

class libp2p.security.insecure.pb.plaintext_pb2.Exchange

Bases: [Message](#), [Message](#)

DESCRIPTOR = <google._upb._message.Descriptor object>

Module contents

Submodules

libp2p.security.insecure.transport module

```
class libp2p.security.insecure.transport.InsecureSession(*, local_peer: ID, local_private_key:
    PrivateKey, remote_peer: ID,
    remote_permanent_pubkey: PublicKey,
    is_initiator: bool, conn:
    ReadWriteCloser)
```

Bases: *BaseSession*

async `close()` → *None*

async `read(n: int | None = None)` → *bytes*

async `write(data: bytes)` → *None*

```
class libp2p.security.insecure.transport.InsecureTransport(local_key_pair:
    ~libp2p.crypto.keys.KeyPair,
    secure_bytes_provider:
    ~typing.Callable[[int], bytes] =
    <function
    default_secure_bytes_provider>)
```

Bases: *BaseSecureTransport*

Provides the “identity” upgrader for a *IRawConnection*, i.e. the upgraded transport does not add any additional security.

async `secure_inbound(conn: IRawConnection)` → *ISecureConn*

Secure the connection, either locally or by communicating with opposing node via *conn*, for an inbound connection (i.e. we are not the initiator)

Returns

secure connection object (that implements *secure_conn_interface*)

async `secure_outbound(conn: IRawConnection, peer_id: ID)` → *ISecureConn*

Secure the connection, either locally or by communicating with opposing node via *conn*, for an inbound connection (i.e. we are the initiator)

Returns

secure connection object (that implements *secure_conn_interface*)

```
class libp2p.security.insecure.transport.PlaintextHandshakeReadWriter(read_write_closer:
    ReadWriteCloser)
```

Bases: *VarIntLengthMsgReadWriter*

max_msg_size: *int* = 65536

`libp2p.security.insecure.transport.make_exchange_message(pubkey: PublicKey)` → *Exchange*

```
async libp2p.security.insecure.transport.run_handshake(local_peer: ID, local_private_key:
    PrivateKey, conn: IRawConnection,
    is_initiator: bool, remote_peer_id: ID) →
    ISecureConn
```

Raise *HandshakeFailure* when handshake failed.

Module contents

libp2p.security.noise package

Subpackages

libp2p.security.noise.pb package

Submodules

libp2p.security.noise.pb.noise_pb2 module

Generated protocol buffer code.

```
class libp2p.security.noise.pb.noise_pb2.NoiseHandshakePayload
```

Bases: `Message`, `Message`

DESCRIPTOR = <google._upb._message.Descriptor object>

Module contents

Submodules

libp2p.security.noise.exceptions module

```
exception libp2p.security.noise.exceptions.HandshakeHasNotFinished
```

Bases: `NoiseFailure`

```
exception libp2p.security.noise.exceptions.InvalidSignature
```

Bases: `NoiseFailure`

```
exception libp2p.security.noise.exceptions.NoiseFailure
```

Bases: `HandshakeFailure`

```
exception libp2p.security.noise.exceptions.NoiseStateError
```

Bases: `NoiseFailure`

Raised when anything goes wrong in the noise state in *noiseprotocol* package.

```
exception libp2p.security.noise.exceptions.PeerIDMismatchesPubkey
```

Bases: `NoiseFailure`

libp2p.security.noise.io module

```
class libp2p.security.noise.io.BaseNoiseMsgReadWriter(conn: IRawConnection, noise_state:
                                                    NoiseConnection)
```

Bases: `EncryptedMsgReadWriter`

The base implementation of noise message reader/writer.

encrypt and *decrypt* are not implemented here, which should be implemented by the subclasses.

`async close() → None`

`noise_state: NoiseConnection`

`prefix: bytes = b'\x00'`

`async read_msg(prefix_encoded: bool = False) → bytes`

`read_writer: MsgReadWriteCloser`

`async write_msg(data: bytes, prefix_encoded: bool = False) → None`

`class libp2p.security.noise.io.NoiseHandshakeReadWriter(conn: IRawConnection, noise_state: NoiseConnection)`

Bases: `BaseNoiseMsgReadWriter`

`decrypt(data: bytes) → bytes`

`encrypt(data: bytes) → bytes`

`class libp2p.security.noise.io.NoisePacketReadWriter(read_write_closer: ReadWriteCloser)`

Bases: `FixedSizeLenMsgReadWriter`

`size_len_bytes: int = 2`

`class libp2p.security.noise.io.NoiseTransportReadWriter(conn: IRawConnection, noise_state: NoiseConnection)`

Bases: `BaseNoiseMsgReadWriter`

`decrypt(data: bytes) → bytes`

`encrypt(data: bytes) → bytes`

libp2p.security.noise.messages module

`class libp2p.security.noise.messages.NoiseHandshakePayload(id_pubkey: libp2p.crypto.keys.PublicKey, id_sig: bytes, early_data: bytes = None)`

Bases: `object`

`classmethod deserialize(protobuf_bytes: bytes) → NoiseHandshakePayload`

`early_data: bytes = None`

`id_pubkey: PublicKey`

`id_sig: bytes`

`serialize() → bytes`

`libp2p.security.noise.messages.make_data_to_be_signed(noise_static_pubkey: PublicKey) → bytes`

`libp2p.security.noise.messages.make_handshake_payload_sig(id_privkey: PrivateKey, noise_static_pubkey: PublicKey) → bytes`

```
libp2p.security.noise.messages.verify_handshake_payload_sig(payload: NoiseHandshakePayload,
                                                            noise_static_pubkey: PublicKey) →
                                                            bool
```

Verify if the signature

1. is composed of the data *SIGNED_DATA_PREFIX* ++ *noise_static_pubkey* and
2. signed by the private key corresponding to *id_pubkey*

libp2p.security.noise.patterns module

```
class libp2p.security.noise.patterns.BasePattern
```

Bases: *IPattern*

create_noise_state() → *NoiseConnection*

early_data: *bytes*

libp2p_privkey: *PrivateKey*

local_peer: *ID*

make_handshake_payload() → *NoiseHandshakePayload*

noise_static_key: *PrivateKey*

protocol_name: *bytes*

```
class libp2p.security.noise.patterns.IPattern
```

Bases: *ABC*

abstract async handshake_inbound(conn: *IRawConnection*) → *ISecureConn*

abstract async handshake_outbound(conn: *IRawConnection*, remote_peer: *ID*) → *ISecureConn*

```
class libp2p.security.noise.patterns.PatternXX(local_peer: ID, libp2p_privkey: PrivateKey,
                                                noise_static_key: PrivateKey, early_data: bytes | None
                                                = None)
```

Bases: *BasePattern*

async handshake_inbound(conn: *IRawConnection*) → *ISecureConn*

async handshake_outbound(conn: *IRawConnection*, remote_peer: *ID*) → *ISecureConn*

libp2p.security.noise.transport module

```
class libp2p.security.noise.transport.Transport(libp2p_keypair: KeyPair, noise_privkey: PrivateKey |
                                                None = None, early_data: bytes | None = None,
                                                with_noise_pipes: bool = False)
```

Bases: *ISecureTransport*

early_data: *bytes*

get_pattern() → *IPattern*

libp2p_privkey: *PrivateKey*

local_peer: *ID*

noise_privkey: *PrivateKey*

async secure_inbound(*conn: IRawConnection*) → *ISecureConn*

Secure the connection, either locally or by communicating with opposing node via conn, for an inbound connection (i.e. we are not the initiator)

Returns

secure connection object (that implements secure_conn_interface)

async secure_outbound(*conn: IRawConnection, peer_id: ID*) → *ISecureConn*

Secure the connection, either locally or by communicating with opposing node via conn, for an inbound connection (i.e. we are the initiator)

Returns

secure connection object (that implements secure_conn_interface)

with_noise_pipes: *bool*

Module contents

libp2p.security.secio package

Subpackages

libp2p.security.secio.pb package

Submodules

libp2p.security.secio.pb.spipe_pb2 module

Generated protocol buffer code.

class libp2p.security.secio.pb.spipe_pb2.**Exchange**

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class libp2p.security.secio.pb.spipe_pb2.**Propose**

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

Module contents

Submodules

libp2p.security.secio.exceptions module

exception libp2p.security.secio.exceptions.IncompatibleChoices

Bases: *SecioException*

exception libp2p.security.secio.exceptions.InconsistentNonce

Bases: *SecioException*

exception libp2p.security.secio.exceptions.InvalidSignatureOnExchange

Bases: *SecioException*

exception libp2p.security.secio.exceptions.PeerMismatchException

Bases: *SecioException*

exception libp2p.security.secio.exceptions.SecioException

Bases: *HandshakeFailure*

exception libp2p.security.secio.exceptions.SedesException

Bases: *SecioException*

exception libp2p.security.secio.exceptions.SelfEncryption

Bases: *SecioException*

Raised to indicate that a host is attempting to encrypt communications with itself.

libp2p.security.secio.transport module

class libp2p.security.secio.transport.EncryptionParameters

Bases: *object*

cipher_type: *str*

curve_type: *str*

ephemeral_public_key: *PublicKey*

hash_type: *str*

permanent_public_key: *PublicKey*

class libp2p.security.secio.transport.Proposal(*nonce: bytes, public_key: PublicKey, exchanges: str = 'P-256', ciphers: str = 'AES-128', hashes: str = 'SHA256'*)

Bases: *object*

Represents the set of session parameters one peer in a pair of peers attempting to negotiate a *secio* channel prefers.

calculate_peer_id() → *ID*

ciphers: *str* = 'AES-128'

```
classmethod deserialize(protobuf_bytes: bytes) → Proposal
exchanges: str = 'P-256'
hashes: str = 'SHA256'
nonce: bytes
public_key: PublicKey
serialize() → bytes

class libp2p.security.secio.transport.SecioMsgReadWriter(local_encryption_parameters:
    EncryptionParameters,
    remote_encryption_parameters:
    EncryptionParameters, read_writer:
    SecioPacketReadWriter)

    Bases: EncryptedMsgReadWriter
    async close() → None
    decrypt(data: bytes) → bytes
    encrypt(data: bytes) → bytes
    async read_msg() → bytes
    read_writer: SecioPacketReadWriter
    async write_msg(msg: bytes) → None

class libp2p.security.secio.transport.SecioPacketReadWriter(read_write_closer: ReadWriteCloser)
    Bases: FixedSizeLenMsgReadWriter
    size_len_bytes: int = 4

class libp2p.security.secio.transport.SessionParameters
    Bases: object
    local_encryption_parameters: EncryptionParameters
    local_peer: ID
    order: int
    remote_encryption_parameters: EncryptionParameters
    remote_peer: ID
    shared_key: bytes

class libp2p.security.secio.transport.Transport(local_key_pair: ~libp2p.crypto.keys.KeyPair,
    secure_bytes_provider: ~typing.Callable[[int], bytes]
    = <function default_secure_bytes_provider>)

    Bases: BaseSecureTransport

    Provide a security upgrader for a IRawConnection, following the secio protocol defined in the libp2p specs.

    get_nonce() → bytes
```


async secure_inbound(*conn*: *IRawConnection*) → *ISecureConn*

Secure the connection, either locally or by communicating with opposing node via *conn*, for an inbound connection (i.e. we are not the initiator)

Returns

secure connection object (that implements *secure_conn_interface*)

async secure_outbound(*conn*: *IRawConnection*, *peer_id*: *ID*) → *ISecureConn*

Secure the connection, either locally or by communicating with opposing node via *conn*, for an inbound connection (i.e. we are the initiator)

Returns

secure connection object (that implements *secure_conn_interface*)

async libp2p.security.secio.transport.create_secure_session(*local_nonce*: *bytes*, *local_peer*: *ID*, *local_private_key*: *PrivateKey*, *conn*: *IRawConnection*, *remote_peer*: *ID* | *None* = *None*) → *ISecureConn*

Attempt the initial *secio* handshake with the remote peer.

If successful, return an object that provides secure communication to the *remote_peer*. Raise *SecioException* when *conn* closed. Raise *InconsistentNonce* when handshake failed

Module contents

Submodules

libp2p.security.base_session module

class libp2p.security.base_session.BaseSession(*, *local_peer*: *ID*, *local_private_key*: *PrivateKey*, *remote_peer*: *ID*, *remote_permanent_pubkey*: *PublicKey*, *is_initiator*: *bool*)

Bases: *ISecureConn*

BaseSession is not fully instantiated from its abstract classes as it is only meant to be used in classes that derive from it.

get_local_peer() → *ID*

get_local_private_key() → *PrivateKey*

get_remote_peer() → *ID*

get_remote_public_key() → *PublicKey* | *None*

local_peer: *ID*

local_private_key: *PrivateKey*

remote_peer: *ID*

remote_permanent_pubkey: *PublicKey*

libp2p.security.base_transport module

```
class libp2p.security.base_transport.BaseSecureTransport(local_key_pair:
    ~libp2p.crypto.keys.KeyPair,
    secure_bytes_provider:
    ~typing.Callable[[int], bytes] = <function
    default_secure_bytes_provider>)
```

Bases: *ISecureTransport*

BaseSecureTransport is not fully instantiated from its abstract classes as it is only meant to be used in classes that derive from it.

Clients can provide a strategy to get cryptographically secure bytes of a given length. A default implementation is provided using the secrets module from the standard library.

```
libp2p.security.base_transport.default_secure_bytes_provider(n: int) → bytes
```

libp2p.security.exceptions module

```
exception libp2p.security.exceptions.HandshakeFailure
```

Bases: *BaseLibp2pError*

libp2p.security.secure_conn_interface module

```
class libp2p.security.secure_conn_interface.AbstractSecureConn
```

Bases: *ABC*

```
abstract get_local_peer() → ID
```

```
abstract get_local_private_key() → PrivateKey
```

```
abstract get_remote_peer() → ID
```

```
abstract get_remote_public_key() → PublicKey
```

```
class libp2p.security.secure_conn_interface.ISecureConn
```

Bases: *AbstractSecureConn, IRawConnection*

libp2p.security.secure_session module

```
class libp2p.security.secure_session.SecureSession(*, local_peer: ID, local_private_key: PrivateKey,
    remote_peer: ID, remote_permanent_pubkey:
    PublicKey, is_initiator: bool, conn:
    EncryptedMsgReadWriter)
```

Bases: *BaseSession*

```
buf: BytesIO
```

```
async close() → None
```

```
high_watermark: int
```

```

low_watermark: int

async read(n: int | None = None) → bytes

async write(data: bytes) → None

```

libp2p.security.secure_transport_interface module

class libp2p.security.secure_transport_interface.ISecureTransport

Bases: ABC

abstract **async** **secure_inbound**(conn: IRawConnection) → ISecureConn

Secure the connection, either locally or by communicating with opposing node via conn, for an inbound connection (i.e. we are not the initiator)

Returns

secure connection object (that implements secure_conn_interface)

abstract **async** **secure_outbound**(conn: IRawConnection, peer_id: ID) → ISecureConn

Secure the connection, either locally or by communicating with opposing node via conn, for an inbound connection (i.e. we are the initiator)

Returns

secure connection object (that implements secure_conn_interface)

libp2p.security.security_multistream module

class libp2p.security.security_multistream.SecurityMultistream(*secure_transports_by_protocol:*
Mapping[TProtocol,
ISecureTransport])

Bases: ABC

SSMuxer is a multistream stream security transport multiplexer.

Go implementation: github.com/libp2p/go-conn-security-multistream/ssms.go

add_transport(protocol: TProtocol, transport: ISecureTransport) → None

Add a protocol and its corresponding transport to multistream- select(multiselect). The order that a protocol is added is exactly the precedence it is negotiated in multiselect.

Parameters

- **protocol** – the protocol name, which is negotiated in multiselect.
- **transport** – the corresponding transportation to the protocol.

multiselect: Multiselect

multiselect_client: MultiselectClient

async **secure_inbound**(conn: IRawConnection) → ISecureConn

Secure the connection, either locally or by communicating with opposing node via conn, for an inbound connection (i.e. we are not the initiator)

Returns

secure connection object (that implements secure_conn_interface)

async secure_outbound(*conn*: [IRawConnection](#), *peer_id*: [ID](#)) → [ISecureConn](#)

Secure the connection, either locally or by communicating with opposing node via conn, for an inbound connection (i.e. we are the initiator)

Returns

secure connection object (that implements `secure_conn_interface`)

async select_transport(*conn*: [IRawConnection](#), *is_initiator*: *bool*) → [ISecureTransport](#)

Select a transport that both us and the node on the other end of conn support and agree on.

Parameters

- **conn** – conn to choose a transport over
- **is_initiator** – true if we are the initiator, false otherwise

Returns

selected secure transport

transports: [OrderedDict](#)[[TProtocol](#), [ISecureTransport](#)]

Module contents

[libp2p.stream_muxer package](#)

[Subpackages](#)

[libp2p.stream_muxer.mplex package](#)

[Submodules](#)

[libp2p.stream_muxer.mplex.constants module](#)

class `libp2p.stream_muxer.mplex.constants.HeaderTags`(*value*)

Bases: [Enum](#)

An enumeration.

CloseInitiator = 4

CloseReceiver = 3

MessageInitiator = 2

MessageReceiver = 1

NewStream = 0

ResetInitiator = 6

ResetReceiver = 5

libp2p.stream_muxer.mplex.datastructures module

class libp2p.stream_muxer.mplex.datastructures.**StreamID**(*channel_id*, *is_initiator*)

Bases: *NamedTuple*

channel_id: *int*

Alias for field number 0

is_initiator: *bool*

Alias for field number 1

libp2p.stream_muxer.mplex.exceptions module

exception libp2p.stream_muxer.mplex.exceptions.**MplexError**

Bases: *MuxedConnError*

exception libp2p.stream_muxer.mplex.exceptions.**MplexStreamClosed**

Bases: *MuxedStreamClosed*

exception libp2p.stream_muxer.mplex.exceptions.**MplexStreamEOF**

Bases: *MuxedStreamEOF*

exception libp2p.stream_muxer.mplex.exceptions.**MplexStreamReset**

Bases: *MuxedStreamReset*

exception libp2p.stream_muxer.mplex.exceptions.**MplexUnavailable**

Bases: *MuxedConnUnavailable*

libp2p.stream_muxer.mplex.mplex module

class libp2p.stream_muxer.mplex.mplex.**Mplex**(*secured_conn*: *ISecureConn*, *peer_id*: *ID*)

Bases: *IMuxedConn*

reference: <https://github.com/libp2p/go-mplex/blob/master/multiplex.go>

async **accept_stream()** → *IMuxedStream*

Accept a muxed stream opened by the other end.

async **close()** → *None*

Close the stream muxer and underlying secured connection.

event_closed: *Event*

event_shutting_down: *Event*

event_started: *Event*

async **handle_incoming()** → *None*

Read a message off of the secured connection and add it to the corresponding message buffer.

property **is_closed:** *bool*

Check connection is fully closed.

Returns

true if successful

property is_initiator: `bool`

If this connection is the initiator.

new_stream_receive_channel: `trio.MemoryReceiveChannel[IMuxedStream]`

new_stream_send_channel: `trio.MemorySendChannel[IMuxedStream]`

next_channel_id: `int`

async open_stream() \rightarrow `IMuxedStream`

Create a new muxed_stream.

Returns

a new MplexStream

peer_id: `ID`

async read_message() \rightarrow `Tuple[int, int, bytes]`

Read a single message off of the secured connection.

Returns

stream_id, flag, message contents

secured_conn: `ISecureConn`

async send_message(flag: HeaderTags, data: bytes | None, stream_id: StreamID) \rightarrow `int`

Send a message over the connection.

Parameters

- **flag** – header to use
- **data** – data to send in the message
- **stream_id** – stream the message is in

async start() \rightarrow `None`

Start the multiplexer.

streams: `Dict[StreamID, MplexStream]`

streams_lock: `Lock`

streams_msg_channels: `Dict[StreamID, trio.MemorySendChannel[bytes]]`

async write_to_stream(_bytes: bytes) \rightarrow `None`

Write a byte array to a secured connection.

Parameters

_bytes – byte array to write

Returns

length written

libp2p.stream_muxer.mplex.mplex_stream module

```
class libp2p.stream_muxer.mplex.mplex_stream.MplexStream(name: str, stream_id: StreamID,
                                                         muxed_conn: Mplex,
                                                         incoming_data_channel:
                                                         trio.MemoryReceiveChannel[bytes])
```

Bases: *IMuxedStream*

reference: <https://github.com/libp2p/go-mplex/blob/master/stream.go>

async **close()** → *None*

Closing a stream closes it for writing and closes the remote end for reading but allows writing in the other direction.

close_lock: *Lock*

event_local_closed: *Event*

event_remote_closed: *Event*

event_reset: *Event*

incoming_data_channel: *trio.MemoryReceiveChannel[bytes]*

property is_initiator: *bool*

muxed_conn: *Mplex*

name: *str*

async **read**(n: *int* | *None* = *None*) → *bytes*

Read up to n bytes. Read possibly returns fewer than n bytes, if there are not enough bytes in the Mplex buffer. If n is None, read until EOF.

Parameters

n – number of bytes to read

Returns

bytes actually read

read_deadline: *int*

async **reset()** → *None*

Close both ends of the stream tells this remote side to hang up.

set_deadline(ttl: *int*) → *bool*

Set deadline for muxed stream.

Returns

True if successful

set_read_deadline(ttl: *int*) → *bool*

Set read deadline for muxed stream.

Returns

True if successful

set_write_deadline(*ttl: int*) → bool
Set write deadline for muxed stream.

Returns
True if successful

stream_id: *StreamID*

async write(*data: bytes*) → None
Write to stream.

Returns
number of bytes written

write_deadline: *int*

Module contents

Submodules

libp2p.stream_muxer.abc module

class libp2p.stream_muxer.abc.**IMuxedConn**(*conn: ISecureConn, peer_id: ID*)
Bases: *ABC*
reference: <https://github.com/libp2p/go-stream-muxer/blob/master/muxer.go>

abstract async accept_stream() → *IMuxedStream*
Accept a muxed stream opened by the other end.

abstract async close() → None
Close connection.

event_started: *Event*

abstract property is_closed: *bool*
Check connection is fully closed.

Returns
true if successful

abstract property is_initiator: *bool*
If this connection is the initiator.

abstract async open_stream() → *IMuxedStream*
Create a new muxed_stream.

Returns
a new *IMuxedStream* stream

peer_id: *ID*

abstract async start() → None
Start the multiplexer.

class libp2p.stream_muxer.abc.**IMuxedStream**
Bases: *ReadWriteCloser*

muxed_conn: *IMuxedConn*

abstract async reset() \rightarrow *None*

Close both ends of the stream tells this remote side to hang up.

abstract set_deadline(ttl: int) \rightarrow *bool*

Set deadline for muxed stream.

Returns

a new stream

libp2p.stream_muxer.exceptions module

exception libp2p.stream_muxer.exceptions.MuxedConnError

Bases: *BaseLibp2pError*

exception libp2p.stream_muxer.exceptions.MuxedConnUnavailable

Bases: *MuxedConnError*

exception libp2p.stream_muxer.exceptions.MuxedStreamClosed

Bases: *MuxedStreamError*

exception libp2p.stream_muxer.exceptions.MuxedStreamEOF

Bases: *MuxedStreamError*, *EOFError*

exception libp2p.stream_muxer.exceptions.MuxedStreamError

Bases: *BaseLibp2pError*

exception libp2p.stream_muxer.exceptions.MuxedStreamReset

Bases: *MuxedStreamError*

libp2p.stream_muxer.muxer_multistream module

class libp2p.stream_muxer.muxer_multistream.MuxerMultistream(*muxer_transports_by_protocol:*
Mapping[TProtocol,
Type[IMuxedConn]])

Bases: *object*

MuxerMultistream is a multistream stream muxed transport multiplexer.

go implementation: github.com/libp2p/go-stream-muxer-multistream/multistream.go

add_transport(protocol: TProtocol, transport: Type[IMuxedConn]) \rightarrow *None*

Add a protocol and its corresponding transport to multistream- select(multiselect). The order that a protocol is added is exactly the precedence it is negotiated in multiselect.

Parameters

- **protocol** – the protocol name, which is negotiated in multiselect.
- **transport** – the corresponding transportation to the protocol.

multiselect: *Multiselect*

multiselect_client: *MultiselectClient*

async new_conn(conn: [ISecureConn](#), peer_id: [ID](#)) → [IMuxedConn](#)

async select_transport(conn: [IRawConnection](#)) → [Type\[IMuxedConn\]](#)

Select a transport that both us and the node on the other end of conn support and agree on.

Parameters

conn – conn to choose a transport over

Returns

selected muxer transport

transports: [OrderedDict](#)[[TProtocol](#), [Type\[IMuxedConn\]](#)]

Module contents

libp2p.tools package

Subpackages

libp2p.tools.pubsub package

Submodules

libp2p.tools.pubsub.dummy_account_node module

class [libp2p.tools.pubsub.dummy_account_node.DummyAccountNode](#)(pubsub: [Pubsub](#))

Bases: [Service](#)

Node which has an internal balance mapping, meant to serve as a dummy crypto blockchain.

There is no actual blockchain, just a simple map indicating how much crypto each user in the mappings holds

classmethod create(number: [int](#)) → [AsyncIterator](#)[[Tuple](#)[[DummyAccountNode](#), ...]]

Create a new [DummyAccountNode](#) and attach a [libp2p](#) node, a [floodsub](#), and a [pubsub](#) instance to this new node.

We use create as this serves as a factory function and allows us to use [async await](#), unlike the [init](#) function

get_balance(user: [str](#)) → [int](#)

Get balance in crypto for a particular user.

Parameters

user – user to get balance for

Returns

balance of user

async handle_incoming_msgs() → [None](#)

Handle all incoming messages on the [CRYPTO_TOPIC](#) from peers.

handle_send_crypto(source_user: [str](#), dest_user: [str](#), amount: [int](#)) → [None](#)

Handle incoming [send_crypto](#) message.

Parameters

- **source_user** – user to send crypto from
- **dest_user** – user to send crypto to

- **amount** – amount of crypto to send

handle_set_crypto(*dest_user: str, amount: int*) → *None*

Handle incoming set_crypto message.

Parameters

- **dest_user** – user to set crypto for
- **amount** – amount of crypto

property host: *IHost*

async publish_send_crypto(*source_user: str, dest_user: str, amount: int*) → *None*

Create a send crypto message and publish that message to all other nodes.

Parameters

- **source_user** – user to send crypto from
- **dest_user** – user to send crypto to
- **amount** – amount of crypto to send

async publish_set_crypto(*user: str, amount: int*) → *None*

Create a set crypto message and publish that message to all other nodes.

Parameters

- **user** – user to set crypto for
- **amount** – amount of crypto

pubsub: *Pubsub*

async run() → *None*

Primary entry point for all service logic.

Note: This method should **not** be directly invoked by user code.

Services may be run using the following approaches.

libp2p.tools.pubsub.floodsub_integration_test_settings module

async libp2p.tools.pubsub.floodsub_integration_test_settings.perform_test_from_obj(*obj, pubsub_factory*) → *None*

Perform pubsub tests from a test object, which is composed as follows:

```
{
  "supported_protocols": ["supported/protocol/1.0.0", ...],
  "adj_list": {
    "node1": ["neighbor1_of_node1", "neighbor2_of_node1", ...],
    "node2": ["neighbor1_of_node2", "neighbor2_of_node2", ...],
    ...
  },
  "topic_map": {
```

(continues on next page)

(continued from previous page)

```
    "topic1": ["node1_subscribed_to_topic1", "node2_subscribed_to_topic1", ...]
  },
  "messages": [
    {
      "topics": ["topic1_for_message", "topic2_for_message", ...],
      "data": b"some contents of the message (newlines are not supported)",
      "node_id": "message sender node id"
    },
    ...
  ]
}
```

Note: In `adj_list`, for any neighbors A and B, only list B as a neighbor of A or B as a neighbor of A once. Do NOT list both A: ["B"] and B:["A"] as the behavior is undefined (even if it may work)

libp2p.tools.pubsub.utils module

async libp2p.tools.pubsub.utils.**connect_some**(*hosts: Sequence[IHost]*, *degree: int*) → None

async libp2p.tools.pubsub.utils.**dense_connect**(*hosts: Sequence[IHost]*) → None

libp2p.tools.pubsub.utils.**make_pubsub_msg**(*origin_id: ID*, *topic_ids: Sequence[str]*, *data: bytes*, *seqno: bytes*) → Message

async libp2p.tools.pubsub.utils.**one_to_all_connect**(*hosts: Sequence[IHost]*, *central_host_index: int*) → None

Module contents

Submodules

libp2p.tools.constants module

class libp2p.tools.constants.**GossipsubParams**(*degree, degree_low, degree_high, time_to_live, gossip_window, gossip_history, heartbeat_initial_delay, heartbeat_interval*)

Bases: `NamedTuple`

degree: `int`

Alias for field number 0

degree_high: `int`

Alias for field number 2

degree_low: `int`

Alias for field number 1

gossip_history: `int`

Alias for field number 5

gossip_window: `int`

Alias for field number 4

heartbeat_initial_delay: `float`

Alias for field number 6

heartbeat_interval: `float`

Alias for field number 7

time_to_live: `int`

Alias for field number 3

libp2p.tools.factories module

class libp2p.tools.factories.DummyRouter

Bases: *IPeerRouting*

async find_peer(peer_id: ID) → *PeerInfo*

Find specific Peer FindPeer searches for a peer with given peer_id, returns a peer.PeerInfo with relevant addresses.

class libp2p.tools.factories.FloodsubFactory(**kwargs)

Bases: Factory

protocols = ('/floodsub/1.0.0',)

class libp2p.tools.factories.GossipsubFactory(**kwargs)

Bases: Factory

degree = 10

degree_high = 11

degree_low = 9

gossip_history = 5

gossip_window = 3

heartbeat_initial_delay = 0.1

heartbeat_interval = 0.5

protocols = ('/meshsub/1.0.0',)

time_to_live = 30

class libp2p.tools.factories.HostFactory(**kwargs)

Bases: Factory

classmethod create_batch_and_listen(number: int, security_protocol: TProtocol = None, muxer_opt: Mapping[TProtocol, Type[IMuxedConn]] = None) → AsyncIterator[Tuple[BasicHost, ...]]

network = <factory.declarations.LazyAttribute object>

```
class libp2p.tools.factories.IDFactory(**kwargs)
    Bases: Factory
    peer_id_bytes = <factory.declarations.LazyFunction object>

class libp2p.tools.factories.PubsubFactory(**kwargs)
    Bases: Factory
    cache_size = None

    classmethod create_and_start(host: IHost, router: IPubsubRouter, cache_size: int, strict_signing: bool,
                                msg_id_constructor: Callable[[Message], bytes] = None) →
                                AsyncIterator[Pubsub]

    classmethod create_batch_with_floodsub(number: int, cache_size: int = None, strict_signing: bool =
                                           False, protocols: ~typing.Sequence[TProtocol] = None,
                                           security_protocol: TProtocol = None, muxer_opt:
                                           ~typing.Mapping[TProtocol,
                                           ~typing.Type[~libp2p.stream_muxer.abc.IMuxedConn]] =
                                           None, msg_id_constructor:
                                           ~typing.Callable[[-libp2p.pubsub.pb.rpc_pb2.Message],
                                           bytes] = <function get_peer_and_seqno_msg_id>) →
                                           AsyncIterator[Tuple[Pubsub, ...]]

    classmethod create_batch_with_gossipsub(number: int, *, cache_size: int = None, strict_signing:
                                           bool = False, protocols: ~typing.Sequence[TProtocol] =
                                           None, degree: int = 10, degree_low: int = 9, degree_high:
                                           int = 11, time_to_live: int = 30, gossip_window: int = 3,
                                           gossip_history: int = 5, heartbeat_interval: float = 0.5,
                                           heartbeat_initial_delay: float = 0.1, security_protocol:
                                           TProtocol = None, muxer_opt:
                                           ~typing.Mapping[TProtocol,
                                           ~typing.Type[~libp2p.stream_muxer.abc.IMuxedConn]] =
                                           None, msg_id_constructor:
                                           ~typing.Callable[[-libp2p.pubsub.pb.rpc_pb2.Message],
                                           bytes] = <function get_peer_and_seqno_msg_id>) →
                                           AsyncIterator[Tuple[Pubsub, ...]]

    host = <factory.declarations.SubFactory object>

    router = None

    strict_signing = False

class libp2p.tools.factories.RoutedHostFactory(**kwargs)
    Bases: Factory

    classmethod create_batch_and_listen(number: int, security_protocol: TProtocol = None, muxer_opt:
                                         Mapping[TProtocol, Type[IMuxedConn]] = None) →
                                         AsyncIterator[Tuple[RoutedHost, ...]]

    network = <factory.declarations.LazyAttribute object>

    router = <factory.declarations.LazyFunction object>

class libp2p.tools.factories.SwarmFactory(**kwargs)
    Bases: Factory
```

```

classmethod create_and_listen(key_pair: KeyPair = None, security_protocol: TProtocol = None,
                               muxer_opt: Mapping[TProtocol, Type[IMuxedConn]] = None) →
                               AsyncIterator[Swarm]

classmethod create_batch_and_listen(number: int, security_protocol: TProtocol = None, muxer_opt:
                                     Mapping[TProtocol, Type[IMuxedConn]] = None) →
                                     AsyncIterator[Tuple[Swarm, ...]]

peer_id = <factory.declarations.LazyAttribute object>
peerstore = <factory.declarations.LazyAttribute object>
transport = <factory.declarations.LazyFunction object>
upgrader = <factory.declarations.LazyAttribute object>

libp2p.tools.factories.default_key_pair_factory() → KeyPair

libp2p.tools.factories.default_muxer_transport_factory() → Mapping[TProtocol,
                                                                    Type[IMuxedConn]]

libp2p.tools.factories.host_pair_factory(security_protocol: TProtocol = None, muxer_opt:
                                           Mapping[TProtocol, Type[IMuxedConn]] = None) →
                                           AsyncIterator[Tuple[BasicHost, BasicHost]]

libp2p.tools.factories.initialize_peerstore_with_our_keypair(self_id: ID, key_pair: KeyPair) →
                                                            PeerStore

libp2p.tools.factories.mplex_conn_pair_factory(security_protocol: TProtocol = None) →
                                                AsyncIterator[Tuple[Mplex, Mplex]]

libp2p.tools.factories.mplex_stream_pair_factory(security_protocol: TProtocol = None) →
                                                  AsyncIterator[Tuple[MplexStream, MplexStream]]

libp2p.tools.factories.mplex_transport_factory() → Mapping[TProtocol, Type[IMuxedConn]]

libp2p.tools.factories.net_stream_pair_factory(security_protocol: TProtocol = None, muxer_opt:
                                                Mapping[TProtocol, Type[IMuxedConn]] = None) →
                                                AsyncIterator[Tuple[INetStream, INetStream]]

libp2p.tools.factories.noise_conn_factory(nursery: Nursery) → AsyncIterator[Tuple[ISecureConn,
                                                                                   ISecureConn]]

libp2p.tools.factories.noise_handshake_payload_factory() → NoiseHandshakePayload

libp2p.tools.factories.noise_static_key_factory() → PrivateKey

libp2p.tools.factories.noise_transport_factory(key_pair: KeyPair) → ISecureTransport

libp2p.tools.factories.plaintext_transport_factory(key_pair: KeyPair) → ISecureTransport

libp2p.tools.factories.raw_conn_factory(nursery: Nursery) → AsyncIterator[Tuple[IRawConnection,
                                                                                   IRawConnection]]

libp2p.tools.factories.secio_transport_factory(key_pair: KeyPair) → ISecureTransport

```

```
libp2p.tools.factories.security_options_factory_factory(protocol_id: TProtocol | None = None) →  
    Callable[[KeyPair], Mapping[TProtocol,  
        ISecureTransport]]
```

```
libp2p.tools.factories.swarm_conn_pair_factory(security_protocol: TProtocol = None, muxer_opt:  
    Mapping[TProtocol, Type[IMuxedConn]] = None) →  
    AsyncIterator[Tuple[SwarmConn, SwarmConn]]
```

```
libp2p.tools.factories.swarm_pair_factory(security_protocol: TProtocol = None, muxer_opt:  
    Mapping[TProtocol, Type[IMuxedConn]] = None) →  
    AsyncIterator[Tuple[Swarm, Swarm]]
```

libp2p.tools.utils module

```
async libp2p.tools.utils.connect(node1: IHost, node2: IHost) → None  
    Connect node1 to node2.
```

```
async libp2p.tools.utils.connect_swarm(swarm_0: Swarm, swarm_1: Swarm) → None
```

```
libp2p.tools.utils.create_echo_stream_handler(ack_prefix: str) → Callable[[INetStream],  
    Awaitable[None]]
```

Module contents

libp2p.transport package

Subpackages

libp2p.transport.tcp package

Submodules

libp2p.transport.tcp.tcp module

```
class libp2p.transport.tcp.tcp.TCP
```

Bases: *ITransport*

```
create_listener(handler_function: Callable[[ReadWriteCloser], Awaitable[None]]) → TCPListener
```

Create listener on transport.

Parameters

handler_function – a function called when a new connection is received that takes a connection as argument which implements interface-connection

Returns

a listener object that implements listener_interface.py

```
async dial(maddr: Multiaddr) → IRawConnection
```

Dial a transport to peer listening on multiaddr.

Parameters

maddr – multiaddr of peer

Returns

RawConnection if successful

Raises

OpenConnectionError – raised when failed to open connection

```
class libp2p.transport.tcp.tcp.TCPListener(handler_function: Callable[[ReadWriteCloser],  
                                         Awaitable[None]])
```

Bases: *IListener*

async `close()` → *None*

get_addrs() → *Tuple*[*Multiaddr*, ...]

Retrieve list of addresses the listener is listening on.

Returns

return list of addrs

async `listen(maddr: Multiaddr, nursery: Nursery)` → *None*

Put listener in listening mode and wait for incoming connections.

Parameters

maddr – maddr of peer

Returns

return True if successful

listeners: *List*[*SocketListener*]

Module contents

Submodules

libp2p.transport.exceptions module

exception `libp2p.transport.exceptions.MuxerUpgradeFailure`

Bases: *UpgradeFailure*

exception `libp2p.transport.exceptions.OpenConnectionError`

Bases: *BaseLibp2pError*

exception `libp2p.transport.exceptions.SecurityUpgradeFailure`

Bases: *UpgradeFailure*

exception `libp2p.transport.exceptions.UpgradeFailure`

Bases: *BaseLibp2pError*

libp2p.transport.listener_interface module

class libp2p.transport.listener_interface.**IListener**

Bases: [ABC](#)

abstract **async** **close()** → [None](#)

abstract **get_addrs()** → [Tuple](#)[[Multiaddr](#), ...]

Retrieve list of addresses the listener is listening on.

Returns

return list of addrs

abstract **async** **listen**(*maddr: Multiaddr*, *nursery: Nursery*) → [bool](#)

Put listener in listening mode and wait for incoming connections.

Parameters

maddr – multiaddr of peer

Returns

return True if successful

libp2p.transport.transport_interface module

class libp2p.transport.transport_interface.**ITransport**

Bases: [ABC](#)

abstract **create_listener**(*handler_function: Callable[[[ReadWriteCloser](#)], [Awaitable](#)[[None](#)]]*) → [IListener](#)

Create listener on transport.

Parameters

handler_function – a function called when a new conntion is received that takes a connection as argument which implements interface-connection

Returns

a listener object that implements listener_interface.py

abstract **async** **dial**(*maddr: Multiaddr*) → [IRawConnection](#)

Dial a transport to peer listening on multiaddr.

Parameters

- **multiaddr** – multiaddr of peer
- **self_id** – peer_id of the dialer (to send to receiver)

Returns

list of multiaddrs

libp2p.transport.typing module

libp2p.transport.upgrader module

```
class libp2p.transport.upgrader.TransportUpgrader(secure_transports_by_protocol:
                                                Mapping[TProtocol, ISecureTransport],
                                                muxer_transports_by_protocol:
                                                Mapping[TProtocol, Type[IMuxedConn]])
```

Bases: `object`

muxer_multistream: `MuxerMultistream`

security_multistream: `SecurityMultistream`

async upgrade_connection(*conn: ISecureConn*, *peer_id: ID*) → *IMuxedConn*
 Upgrade secured connection to a muxed connection.

upgrade_listener(*transport: ITransport*, *listeners: IListener*) → *None*
 Upgrade multiaddr listeners to libp2p-transport listeners.

async upgrade_security(*raw_conn: IRawConnection*, *peer_id: ID*, *is_initiator: bool*) → *ISecureConn*
 Upgrade conn to a secured connection.

Module contents

1.5.2 Submodules

1.5.3 libp2p.exceptions module

exception libp2p.exceptions.BaseLibp2pError
 Bases: `Exception`

exception libp2p.exceptions.MultiError
 Bases: `BaseLibp2pError`
 Raised with multiple exceptions.

exception libp2p.exceptions.ParseError
 Bases: `BaseLibp2pError`

exception libp2p.exceptions.ValidationError
 Bases: `BaseLibp2pError`
 Raised when something does not pass a validation check.

1.5.4 libp2p.typing module

1.5.5 libp2p.utils module

`async libp2p.utils.decode_uvarint_from_stream(reader: Reader) → int`

<https://en.wikipedia.org/wiki/LEB128>.

`libp2p.utils.encode_delim(msg: bytes) → bytes`

`libp2p.utils.encode_uvarint(number: int) → bytes`

Pack *number* into varint bytes.

`libp2p.utils.encode_varint_prefix(msg_bytes: bytes) → bytes`

`async libp2p.utils.read_delim(reader: Reader) → bytes`

`async libp2p.utils.read_varint_prefix_bytes(reader: Reader) → bytes`

1.5.6 Module contents

`libp2p.generate_new_rsa_identity() → KeyPair`

`libp2p.generate_peer_id_from(key_pair: KeyPair) → ID`

`libp2p.new_host(key_pair: KeyPair | None = None, muxer_opt: Mapping[TProtocol, Type[IMuxedConn]] | None = None, sec_opt: Mapping[TProtocol, ISecureTransport] | None = None, peerstore_opt: IPeerStore | None = None, disc_opt: IPeerRouting | None = None) → IHost`

Create a new libp2p host based on the given parameters.

Parameters

- **key_pair** – optional choice of the KeyPair
- **muxer_opt** – optional choice of stream muxer
- **sec_opt** – optional choice of security upgrade
- **peerstore_opt** – optional peerstore
- **disc_opt** – optional discovery

Returns

return a host instance

`libp2p.new_swarm(key_pair: KeyPair | None = None, muxer_opt: Mapping[TProtocol, Type[IMuxedConn]] | None = None, sec_opt: Mapping[TProtocol, ISecureTransport] | None = None, peerstore_opt: IPeerStore | None = None) → INetworkService`

Create a swarm instance based on the parameters.

Parameters

- **key_pair** – optional choice of the KeyPair
- **muxer_opt** – optional choice of stream muxer
- **sec_opt** – optional choice of security upgrade
- **peerstore_opt** – optional peerstore

Returns

return a default swarm instance

1.6 Contributing

1.6.1 Development

To get started, fork the repository to your own github account, then clone it to your development machine:

```
git clone git@github.com:your-github-username/py-libp2p.git
```

then install the development dependencies. We recommend using a virtual environment, such as [virtualenv](#).

```
cd py-libp2p
virtualenv -p python venv
. venv/bin/activate
python -m pip install -e ".[dev]"
pre-commit install
```

Dependencies

On Debian Linux you will need to ensure that you have the [GNU Multiprecision Arithmetic Library](#) installed since it is a dependency of the fastecdsa package. You can install it using the following command:

```
sudo apt-get install libgmp-dev
```

Requirements

The protobuf description in this repository was generated by [protoc](#) at version 25.3.

1.6.2 Testing

Running that tests is a great way to explore the codebase.

You can run all the tests with `pytest tests`.

At this time, the interop tests are not passing. You can run just the internal tests with `pytest tests/core`.

1.6.3 Code Style

We use [pre-commit](#) to maintain consistent code style. Once installed, it will run automatically with every commit. You can also run it manually with:

```
make lint
```

If you need to make a commit that skips the pre-commit checks, you can do so with `git commit --no-verify`.

This project uses [mypy](#) for static type checking, though it is not yet complete. All new code should be fully typed, and we are working to add types to the existing codebase.

1.6.4 Releasing

Releases are typically done from the `main` branch, except when releasing a beta (in which case the beta is released from `main`, and the previous stable branch is released from said branch).

Final test before each release

Before releasing a new version, build and test the package that will be released:

```
git checkout main && git pull
make package-test
```

This will build the package and install it in a temporary virtual environment. Follow the instructions to activate the venv and test whatever you think is important.

You can also preview the release notes:

```
towncrier --draft
```

Build the release notes

Before bumping the version number, build the release notes. You must include the part of the version to bump (see below), which changes how the version number will show in the release notes.

```
make notes bump=$$VERSION_PART_TO_BUMP$$
```

If there are any errors, be sure to re-run `make notes` until it works.

Push the release to github & pypi

After confirming that the release package looks okay, release a new version:

```
make release bump=$$VERSION_PART_TO_BUMP$$
```

This command will:

- Bump the version number as specified in `.pyproject.toml` and `setup.py`.
- Create a git commit and tag for the new version.
- Build the package.
- Push the commit and tag to github.
- Push the new package files to pypi.

Which version part to bump

`$$VERSION_PART_TO_BUMP$$` must be one of: `major`, `minor`, `patch`, `stage`, or `devnum`.

The version format for this repo is `{major}.{minor}.{patch}` for stable, and `{major}.{minor}.{patch}-{stage}.{devnum}` for unstable (stage can be alpha or beta).

If you are in a beta version, `make release bump=stage` will switch to a stable.

To issue an unstable version when the current version is stable, specify the new version explicitly, like `make release bump="--new-version 4.0.0-alpha.1"`

You can see what the result of bumping any particular version part would be with `bump-my-version show-bump`

1.7 History

Prior to 2023, this project was graciously sponsored by the Ethereum Foundation through [Wave 5 of their Grants Program](#).

The creators and original maintainers of this project are:

- [@zixuanzh](#)
- [@alexh](#)
- [@stuckinaboot](#)
- [@robzajac](#)
- [@carver](#)

1.8 Code of Conduct

The libp2p project operates under the [IPFS Code of Conduct](#)

tl;dr:

- Be respectful.
- We're here to help: abuse@ipfs.io
- Abusive behavior is never tolerated.
- Violations of this code may result in swift and permanent expulsion from the IPFS [and libp2p] community.
- "Too long, didn't read" is not a valid excuse for not knowing what is in this document.

PYTHON MODULE INDEX

e

examples, 5

examples.chat, 5

examples.chat.chat, 5

|

libp2p, 80

libp2p.crypto, 11

libp2p.crypto.authenticated_encryption, 6

libp2p.crypto.ecc, 7

libp2p.crypto.ed25519, 8

libp2p.crypto.exceptions, 8

libp2p.crypto.key_exchange, 8

libp2p.crypto.keys, 9

libp2p.crypto.pb, 6

libp2p.crypto.pb.crypto_pb2, 6

libp2p.crypto.rsa, 10

libp2p.crypto.secp256k1, 10

libp2p.crypto.serialization, 11

libp2p.exceptions, 79

libp2p.host, 15

libp2p.host.basic_host, 11

libp2p.host.defaults, 13

libp2p.host.exceptions, 13

libp2p.host.host_interface, 13

libp2p.host.ping, 14

libp2p.host.routed_host, 15

libp2p.identity, 16

libp2p.identity.identify, 16

libp2p.identity.identify.pb, 15

libp2p.identity.identify.pb.identify_pb2, 15

libp2p.identity.identify.protocol, 15

libp2p.io, 18

libp2p.io.abc, 16

libp2p.io.exceptions, 17

libp2p.io.msgio, 17

libp2p.io.trio, 18

libp2p.io.utils, 18

libp2p.network, 26

libp2p.network.connection, 20

libp2p.network.connection.exceptions, 18

libp2p.network.connection.net_connection_interface,
19

libp2p.network.connection.raw_connection, 19

libp2p.network.connection.raw_connection_interface,
19

libp2p.network.connection.swarm_connection,
20

libp2p.network.exceptions, 22

libp2p.network.network_interface, 22

libp2p.network.notifee_interface, 23

libp2p.network.stream, 22

libp2p.network.stream.exceptions, 20

libp2p.network.stream.net_stream, 21

libp2p.network.stream.net_stream_interface,
21

libp2p.network.swarm, 24

libp2p.peer, 36

libp2p.peer.addrbook_interface, 26

libp2p.peer.id, 27

libp2p.peer.peerdata, 27

libp2p.peer.peerdata_interface, 29

libp2p.peer.peerinfo, 30

libp2p.peer.peermetadata_interface, 31

libp2p.peer.peerstore, 31

libp2p.peer.peerstore_interface, 34

libp2p.protocol_muxer, 41

libp2p.protocol_muxer.exceptions, 36

libp2p.protocol_muxer.multiselect, 37

libp2p.protocol_muxer.multiselect_client, 38

libp2p.protocol_muxer.multiselect_client_interface,
39

libp2p.protocol_muxer.multiselect_communicator,
39

libp2p.protocol_muxer.multiselect_communicator_interface,
40

libp2p.protocol_muxer.multiselect_muxer_interface,
40

libp2p.pubsub, 53

libp2p.pubsub.abc, 42

libp2p.pubsub.exceptions, 43

libp2p.pubsub.floodsub, 44

libp2p.pubsub.gossipsub, 45

- libp2p.pubsub.mcache, 47
- libp2p.pubsub.pb, 42
- libp2p.pubsub.pb.rpc_pb2, 41
- libp2p.pubsub.pubsub, 48
- libp2p.pubsub.pubsub_notiffee, 51
- libp2p.pubsub.subscription, 52
- libp2p.pubsub.typing, 52
- libp2p.pubsub.validators, 52
- libp2p.routing, 53
- libp2p.routing.interfaces, 53
- libp2p.security, 64
- libp2p.security.base_session, 61
- libp2p.security.base_transport, 62
- libp2p.security.exceptions, 62
- libp2p.security.insecure, 55
- libp2p.security.insecure.pb, 54
- libp2p.security.insecure.pb.plaintext_pb2, 53
- libp2p.security.insecure.transport, 54
- libp2p.security.noise, 58
- libp2p.security.noise.exceptions, 55
- libp2p.security.noise.io, 55
- libp2p.security.noise.messages, 56
- libp2p.security.noise.patterns, 57
- libp2p.security.noise.pb, 55
- libp2p.security.noise.pb.noise_pb2, 55
- libp2p.security.noise.transport, 57
- libp2p.security.secio, 61
- libp2p.security.secio.exceptions, 59
- libp2p.security.secio.pb, 59
- libp2p.security.secio.pb.spice_pb2, 58
- libp2p.security.secio.transport, 59
- libp2p.security.secure_conn_interface, 62
- libp2p.security.secure_session, 62
- libp2p.security.secure_transport_interface, 63
- libp2p.security.security_multistream, 63
- libp2p.stream_muxer, 70
- libp2p.stream_muxer.abc, 68
- libp2p.stream_muxer.exceptions, 69
- libp2p.stream_muxer.mplex, 68
- libp2p.stream_muxer.mplex.constants, 64
- libp2p.stream_muxer.mplex.datastructures, 65
- libp2p.stream_muxer.mplex.exceptions, 65
- libp2p.stream_muxer.mplex.mplex, 65
- libp2p.stream_muxer.mplex.mplex_stream, 67
- libp2p.stream_muxer.muxer_multistream, 69
- libp2p.tools, 76
- libp2p.tools.constants, 72
- libp2p.tools.factories, 73
- libp2p.tools.pubsub, 72
- libp2p.tools.pubsub.dummy_account_node, 70
- libp2p.tools.pubsub.floodsub_integration_test_settings, 71
- libp2p.tools.pubsub.utils, 72
- libp2p.tools.utils, 76
- libp2p.transport, 79
- libp2p.transport.exceptions, 77
- libp2p.transport.listener_interface, 78
- libp2p.transport.tcp, 77
- libp2p.transport.tcp.tcp, 76
- libp2p.transport.transport_interface, 78
- libp2p.transport.typing, 79
- libp2p.transport.upgrader, 79
- libp2p.typing, 80
- libp2p.utils, 80

INDEX

A

`AbstractSecureConn` (class in `libp2p.security.secure_conn_interface`), 62

`accept_stream()` (`libp2p.stream_muxer.abc.IMuxedConn` method), 68

`accept_stream()` (`libp2p.stream_muxer.mplex.mplex.Mplex` method), 65

`add_addr()` (`libp2p.peer.addrbook_interface.IAddrBook` method), 26

`add_addr()` (`libp2p.peer.peerstore.PeerStore` method), 31

`add_addr()` (`libp2p.peer.peerstore_interface.IPeerStore` method), 34

`add_addrs()` (`libp2p.peer.addrbook_interface.IAddrBook` method), 26

`add_addrs()` (`libp2p.peer.peerdata.PeerData` method), 27

`add_addrs()` (`libp2p.peer.peerdata_interface.IPeerData` method), 29

`add_addrs()` (`libp2p.peer.peerstore.PeerStore` method), 31

`add_addrs()` (`libp2p.peer.peerstore_interface.IPeerStore` method), 34

`add_conn()` (`libp2p.network.swarm.Swarm` method), 24

`add_handler()` (`libp2p.protocol_muxer.multiselect.Multiselect` method), 37

`add_handler()` (`libp2p.protocol_muxer.multiselect_muxer.MultiselectMuxer` method), 40

`add_key_pair()` (`libp2p.peer.peerstore.PeerStore` method), 31

`add_key_pair()` (`libp2p.peer.peerstore_interface.IPeerStore` method), 34

`add_peer()` (`libp2p.pubsub.abc.IPubsubRouter` method), 42

`add_peer()` (`libp2p.pubsub.floodsub.FloodSub` method), 44

`add_peer()` (`libp2p.pubsub.gossipsub.GossipSub` method), 45

`add_privkey()` (`libp2p.peer.peerdata.PeerData` method), 27

`add_privkey()` (`libp2p.peer.peerdata_interface.IPeerData` method), 29

`add_privkey()` (`libp2p.peer.peerstore.PeerStore` method), 31

`add_privkey()` (`libp2p.peer.peerstore_interface.IPeerStore` method), 34

`add_protocols()` (`libp2p.peer.peerdata.PeerData` method), 28

`add_protocols()` (`libp2p.peer.peerdata_interface.IPeerData` method), 29

`add_protocols()` (`libp2p.peer.peerstore.PeerStore` method), 32

`add_protocols()` (`libp2p.peer.peerstore_interface.IPeerStore` method), 34

`add_pubkey()` (`libp2p.peer.peerdata.PeerData` method), 28

`add_pubkey()` (`libp2p.peer.peerdata_interface.IPeerData` method), 29

`add_pubkey()` (`libp2p.peer.peerstore.PeerStore` method), 32

`add_pubkey()` (`libp2p.peer.peerstore_interface.IPeerStore` method), 34

`add_transport()` (`libp2p.security.security_multistream.SecurityMultistream` method), 63

`add_transport()` (`libp2p.stream_muxer.muxer_multistream.MuxerMultistream` method), 69

`addrs` (`libp2p.peer.peerdata.PeerData` attribute), 28

`addrs` (`libp2p.peer.peerinfo.PeerInfo` attribute), 30

`addrs()` (`libp2p.peer.addrbook_interface.IAddrBook` method), 26

`addrs()` (`libp2p.peer.peerstore.PeerStore` method), 32

`addrs()` (`libp2p.peer.peerstore_interface.IPeerStore` method), 35

`attach()` (`libp2p.pubsub.abc.IPubsubRouter` method), 42

`attach()` (`libp2p.pubsub.floodsub.FloodSub` method), 44

`attach()` (`libp2p.pubsub.gossipsub.GossipSub` method), 45

`authenticate()` (`libp2p.crypto.authenticated_encryption.MacAndCipher` method), 6

B

`BaseLibp2pError`, 79

BaseMsgReadWriter (class in libp2p.io.msgio), 17
BaseNoiseMsgReadWriter (class in libp2p.security.noise.io), 55
BasePattern (class in libp2p.security.noise.patterns), 57
BaseSecureTransport (class in libp2p.security.base_transport), 62
BaseSession (class in libp2p.security.base_session), 61
BaseSubscriptionAPI (class in libp2p.pubsub.subscription), 52
BasicHost (class in libp2p.host.basic_host), 11
buf (libp2p.security.secure_session.SecureSession attribute), 62

C

cache_size (libp2p.tools.factories.PubsubFactory attribute), 74
CacheEntry (class in libp2p.pubsub.mcache), 47
calculate_peer_id() (libp2p.security.secio.transport.Proposal method), 59
channel_id (libp2p.stream_muxer.mplex.datastructures.StreamID attribute), 65
cipher_key (libp2p.crypto.authenticated_encryption.EncryptionParameters attribute), 6
cipher_type (libp2p.crypto.authenticated_encryption.EncryptionParameters attribute), 6
cipher_type (libp2p.security.secio.transport.EncryptionParameters attribute), 59
ciphers (libp2p.security.secio.transport.Proposal attribute), 59
clear_addrs() (libp2p.peer.addrbook_interface.IAddrBook method), 27
clear_addrs() (libp2p.peer.peerdata.PeerData method), 28
clear_addrs() (libp2p.peer.peerdata_interface.IPeerData method), 29
clear_addrs() (libp2p.peer.peerstore.PeerStore method), 32
clear_addrs() (libp2p.peer.peerstore_interface.IPeerStore method), 35
close() (libp2p.host.basic_host.BasicHost method), 11
close() (libp2p.host.host_interface.IHost method), 13
close() (libp2p.io.abc.Closer method), 16
close() (libp2p.io.msgio.BaseMsgReadWriter method), 17
close() (libp2p.io.trio.TrioTCPStream method), 18
close() (libp2p.network.connection.raw_connection.RawConnection method), 19
close() (libp2p.network.connection.swarm_connection.SwarmConn method), 20
close() (libp2p.network.network_interface.INetwork method), 22
close() (libp2p.network.stream.net_stream.NetStream method), 21
close() (libp2p.network.swarm.Swarm method), 24
close() (libp2p.security.insecure.transport.InsecureSession method), 54
close() (libp2p.security.noise.io.BaseNoiseMsgReadWriter method), 55
close() (libp2p.security.secio.transport.SecioMsgReadWriter method), 60
close() (libp2p.security.secure_session.SecureSession method), 62
close() (libp2p.stream_muxer.abc.IMuxedConn method), 68
close() (libp2p.stream_muxer.mplex.mplex.Mplex method), 65
close() (libp2p.stream_muxer.mplex.mplex_stream.MplexStream method), 67
close() (libp2p.transport.listener_interface.IListener method), 78
close() (libp2p.transport.tcp.tcp.TCPListener method), 77
close_lock (libp2p.stream_muxer.mplex.mplex_stream.MplexStream attribute), 67
close_peer() (libp2p.network.network_interface.INetwork method), 22
close_peer() (libp2p.network.swarm.Swarm method), 24
closed_stream() (libp2p.network.notifee_interface.INotifee method), 23
closed_stream() (libp2p.pubsub.pubsub_notifee.PubsubNotifee method), 51
closeInitiator (libp2p.stream_muxer.mplex.constants.HeaderTags attribute), 64
Closer (class in libp2p.io.abc), 16
CloseReceiver (libp2p.stream_muxer.mplex.constants.HeaderTags attribute), 64
common_stream_handler (libp2p.network.swarm.Swarm attribute), 24
connect() (in module libp2p.tools.utils), 76
connect() (libp2p.host.basic_host.BasicHost method), 11
connect() (libp2p.host.host_interface.IHost method), 13
connect() (libp2p.host.routed_host.RoutedHost method), 15
connect_some() (in module libp2p.tools.pubsub.utils), 72
connect_swarm() (in module libp2p.tools.utils), 76
connected() (libp2p.network.notifee_interface.INotifee method), 23
connected() (libp2p.pubsub.pubsub_notifee.PubsubNotifee method), 51
ConnectionFailure, 13

[connections](#) (*libp2p.network.network_interface.INetwork* attribute), 22
[connections](#) (*libp2p.network.swarm.Swarm* attribute), 24
[continuously_read_stream\(\)](#) (*libp2p.pubsub.pubsub.Pubsub* method), 48
[ControlGraft](#) (class in *libp2p.pubsub.pb.rpc_pb2*), 41
[ControlIHave](#) (class in *libp2p.pubsub.pb.rpc_pb2*), 41
[ControlIWant](#) (class in *libp2p.pubsub.pb.rpc_pb2*), 41
[ControlMessage](#) (class in *libp2p.pubsub.pb.rpc_pb2*), 41
[ControlPrune](#) (class in *libp2p.pubsub.pb.rpc_pb2*), 41
[counter](#) (*libp2p.pubsub.pubsub.Pubsub* attribute), 48
[create\(\)](#) (*libp2p.tools.pubsub.dummy_account_node.DummyAccountNode* class method), 70
[create_and_listen\(\)](#) (*libp2p.tools.factories.SwarmFactory* class method), 74
[create_and_start\(\)](#) (*libp2p.tools.factories.PubsubFactory* class method), 74
[create_batch_and_listen\(\)](#) (*libp2p.tools.factories.HostFactory* class method), 73
[create_batch_and_listen\(\)](#) (*libp2p.tools.factories.RoutedHostFactory* class method), 74
[create_batch_and_listen\(\)](#) (*libp2p.tools.factories.SwarmFactory* class method), 75
[create_batch_with_floodsub\(\)](#) (*libp2p.tools.factories.PubsubFactory* class method), 74
[create_batch_with_gossipsub\(\)](#) (*libp2p.tools.factories.PubsubFactory* class method), 74
[create_default_stream_handler\(\)](#) (in module *libp2p.network.swarm*), 26
[create_echo_stream_handler\(\)](#) (in module *libp2p.tools.utils*), 76
[create_ephemeral_key_pair\(\)](#) (in module *libp2p.crypto.key_exchange*), 8
[create_listener\(\)](#) (*libp2p.transport.tcp.tcp.TCP* method), 76
[create_listener\(\)](#) (*libp2p.transport.transport_interface.ITransport* method), 78
[create_new_key_pair\(\)](#) (in module *libp2p.crypto.ecc*), 7
[create_new_key_pair\(\)](#) (in module *libp2p.crypto.ed25519*), 8
[create_new_key_pair\(\)](#) (in module *libp2p.crypto.rsa*), 10
[create_new_key_pair\(\)](#) (in module *libp2p.crypto.secp256k1*), 11
[create_noise_state\(\)](#) (*libp2p.security.noise.patterns.BasePattern* method), 57
[create_secure_session\(\)](#) (in module *libp2p.security.secio.transport*), 61
[CryptographyError](#), 8
[curve_type](#) (*libp2p.security.secio.transport.EncryptionParameters* attribute), 59

D

[dead_peer_receive_channel](#) (*libp2p.pubsub.pubsub.Pubsub* attribute), 48
[dead_peers_queue](#) (*libp2p.pubsub.pubsub_notifiee.PubsubNotifiee* attribute), 51
[decode_uvarint_from_stream\(\)](#) (in module *libp2p.utils*), 80
[decrypt\(\)](#) (*libp2p.io.abc.Encrypter* method), 16
[decrypt\(\)](#) (*libp2p.security.noise.io.NoiseHandshakeReadWriter* method), 56
[decrypt\(\)](#) (*libp2p.security.noise.io.NoiseTransportReadWriter* method), 56
[decrypt\(\)](#) (*libp2p.security.secio.transport.SecioMsgReadWriter* method), 60
[decrypt_if_valid\(\)](#) (*libp2p.crypto.authenticated_encryption.MacAndCiphertext* method), 6
[DecryptionFailedException](#), 17
[default_key_pair_factory\(\)](#) (in module *libp2p.tools.factories*), 75
[default_muxer_transport_factory\(\)](#) (in module *libp2p.tools.factories*), 75
[default_secure_bytes_provider\(\)](#) (in module *libp2p.security.base_transport*), 62
[degree](#) (*libp2p.pubsub.gossipsub.GossipSub* attribute), 45
[degree](#) (*libp2p.tools.constants.GossipsubParams* attribute), 72
[degree](#) (*libp2p.tools.factories.GossipsubFactory* attribute), 73
[degree_high](#) (*libp2p.pubsub.gossipsub.GossipSub* attribute), 45
[degree_high](#) (*libp2p.tools.constants.GossipsubParams* attribute), 72
[degree_high](#) (*libp2p.tools.factories.GossipsubFactory* attribute), 73
[degree_low](#) (*libp2p.pubsub.gossipsub.GossipSub* attribute), 45
[degree_low](#) (*libp2p.tools.constants.GossipsubParams* attribute), 72
[degree_low](#) (*libp2p.tools.factories.GossipsubFactory* attribute), 73
[dense_connect\(\)](#) (in module *libp2p.tools.pubsub.utils*), 72

DESCRIPTOR (*libp2p.crypto.pb.crypto_pb2.PrivateKey attribute*), 6

DESCRIPTOR (*libp2p.crypto.pb.crypto_pb2.PublicKey attribute*), 6

DESCRIPTOR (*libp2p.identity.identify.pb.identify_pb2.Identity attribute*), 15

DESCRIPTOR (*libp2p.pubsub.pb.rpc_pb2.ControlGraft attribute*), 41

DESCRIPTOR (*libp2p.pubsub.pb.rpc_pb2.ControlIHave attribute*), 41

DESCRIPTOR (*libp2p.pubsub.pb.rpc_pb2.ControlIWant attribute*), 41

DESCRIPTOR (*libp2p.pubsub.pb.rpc_pb2.ControlMessage attribute*), 41

DESCRIPTOR (*libp2p.pubsub.pb.rpc_pb2.ControlPrune attribute*), 41

DESCRIPTOR (*libp2p.pubsub.pb.rpc_pb2.Message attribute*), 41

DESCRIPTOR (*libp2p.pubsub.pb.rpc_pb2.RPC attribute*), 41

DESCRIPTOR (*libp2p.pubsub.pb.rpc_pb2.RPC.SubOpts attribute*), 41

DESCRIPTOR (*libp2p.pubsub.pb.rpc_pb2.TopicDescriptor attribute*), 42

DESCRIPTOR (*libp2p.pubsub.pb.rpc_pb2.TopicDescriptor.AuthOpts attribute*), 42

DESCRIPTOR (*libp2p.pubsub.pb.rpc_pb2.TopicDescriptor.EncOpts attribute*), 42

DESCRIPTOR (*libp2p.security.insecure.pb.plaintext_pb2.Exchange attribute*), 53

DESCRIPTOR (*libp2p.security.noise.pb.noise_pb2.NoiseHandshakePayload attribute*), 55

DESCRIPTOR (*libp2p.security.secio.pb.spipe_pb2.Exchange attribute*), 58

DESCRIPTOR (*libp2p.security.secio.pb.spipe_pb2.Propose attribute*), 58

deserialize() (*libp2p.crypto.secp256k1.Secp256k1PrivateKey class method*), 10

deserialize() (*libp2p.crypto.secp256k1.Secp256k1PublicKey class method*), 11

deserialize() (*libp2p.security.noise.messages.NoiseHandshakePayload class method*), 56

deserialize() (*libp2p.security.secio.transport.Proposal class method*), 59

deserialize_from_protobuf() (*libp2p.crypto.keys.PrivateKey class method*), 9

deserialize_from_protobuf() (*libp2p.crypto.keys.PublicKey class method*), 9

deserialize_private_key() (*in module libp2p.crypto.serialization*), 11

deserialize_public_key() (*in module libp2p.crypto.serialization*), 11

dial() (*libp2p.transport.tcp.tcp.TCP method*), 76

dial() (*libp2p.transport.transport_interface.ITransport method*), 78

dial_addr() (*libp2p.network.swarm.Swarm method*), 24

dial_peer() (*libp2p.network.network_interface.INetwork method*), 22

dial_peer() (*libp2p.network.swarm.Swarm method*), 24

digest() (*libp2p.peer.id.IdentityHash method*), 27

disconnect() (*libp2p.host.basic_host.BasicHost method*), 11

disconnect() (*libp2p.host.host_interface.IHost method*), 13

disconnected() (*libp2p.network.notifee_interface.INotifee method*), 23

disconnected() (*libp2p.pubsub.pubsub_notifee.PubsubNotifee method*), 51

DummyAccountNode (*class in libp2p.tools.pubsub.dummy_account_node*), 70

DummyRouter (*class in libp2p.tools.factories*), 73

E

early_data (*libp2p.security.noise.messages.NoiseHandshakePayload attribute*), 56

early_data (*libp2p.security.noise.patterns.BasePattern attribute*), 57

early_data (*libp2p.security.noise.transport.Transport attribute*), 57

ecdsa256 (*libp2p.crypto.keys.KeyType attribute*), 9

ECCPrivateKey (*class in libp2p.crypto.ecc*), 7

ECCPublicKey (*class in libp2p.crypto.ecc*), 7

ECDSA (*libp2p.crypto.keys.KeyType attribute*), 9

Ed25519 (*libp2p.crypto.keys.KeyType attribute*), 9

Ed25519PrivateKey (*class in libp2p.crypto.ed25519*), 8

Ed25519PublicKey (*class in libp2p.crypto.ed25519*), 8

emit_control_message() (*libp2p.pubsub.gossipsub.GossipSub method*), 45

emit_graft() (*libp2p.pubsub.gossipsub.GossipSub method*), 45

emit_ihave() (*libp2p.pubsub.gossipsub.GossipSub method*), 45

emit_iwant() (*libp2p.pubsub.gossipsub.GossipSub method*), 45

emit_prune() (*libp2p.pubsub.gossipsub.GossipSub method*), 45

encode_delim() (*in module libp2p.utils*), 80

encode_msg() (*libp2p.io.msgio.BaseMsgReadWriter method*), 17

encode_msg() (*libp2p.io.msgio.FixedSizeLenMsgReadWriter method*), 17

encode_msg() (*libp2p.io.msgio.VarIntLengthMsgReadWriter method*), 18

`encode_msg_with_length()` (in module `libp2p.io.msgio`), 18
`encode_uvarint()` (in module `libp2p.utils`), 80
`encode_varint_prefixed()` (in module `libp2p.utils`), 80
`encrypt()` (`libp2p.crypto.authenticated_encryption.MacAuthChange` (class in `libp2p.security.insecure.pb.plaintext_pb2`), method), 6
`encrypt()` (`libp2p.io.abc.Encrypter` method), 16
`encrypt()` (`libp2p.security.noise.io.NoiseHandshakeReadWriter` method), 56
`encrypt()` (`libp2p.security.noise.io.NoiseTransportReadWriter` method), 56
`encrypt()` (`libp2p.security.secio.transport.SecioMsgReadWriter` method), 60
`EncryptedMsgReadWriter` (class in `libp2p.io.abc`), 16
`Encrypter` (class in `libp2p.io.abc`), 16
`EncryptionParameters` (class in `libp2p.crypto.authenticated_encryption`), 6
`EncryptionParameters` (class in `libp2p.security.secio.transport`), 59
`ephemeral_public_key` (`libp2p.security.secio.transport.EncryptionParameters` attribute), 59
`event_closed` (`libp2p.network.connection.swarm_connection.SwarmConn` attribute), 20
`event_closed` (`libp2p.stream_muxer.mplex.mplex.Mplex` attribute), 65
`event_handle_dead_peer_queue_started` (`libp2p.pubsub.pubsub.Pubsub` attribute), 48
`event_handle_peer_queue_started` (`libp2p.pubsub.pubsub.Pubsub` attribute), 48
`event_listener_nursery_created` (`libp2p.network.swarm.Swarm` attribute), 24
`event_local_closed` (`libp2p.stream_muxer.mplex.mplex_stream.MplexStream` attribute), 67
`event_remote_closed` (`libp2p.stream_muxer.mplex.mplex_stream.MplexStream` attribute), 67
`event_reset` (`libp2p.stream_muxer.mplex.mplex_stream.MplexStream` attribute), 67
`event_shutting_down` (`libp2p.stream_muxer.mplex.mplex.Mplex` attribute), 65
`event_started` (`libp2p.network.connection.net_connection.InterfaceINetConn` attribute), 19
`event_started` (`libp2p.stream_muxer.abc.IMuxedConn` attribute), 68
`event_started` (`libp2p.stream_muxer.mplex.mplex.Mplex` attribute), 65
`examples`

`examples.chat` module, 5
`examples.chat.chat` module, 5
`Exchange` (class in `libp2p.security.insecure.pb.plaintext_pb2`), 53
`Exchange` (class in `libp2p.security.secio.pb.spiffe_pb2`), 58
`exchanges` (`libp2p.security.secio.transport.Proposal` attribute), 60
F
`fanout` (`libp2p.pubsub.gossipsub.GossipSub` attribute), 45
`fanout_heartbeat()` (`libp2p.pubsub.gossipsub.GossipSub` method), 45
`find_peer()` (`libp2p.routing.interfaces.IPeerRouting` method), 53
`find_peer()` (`libp2p.tools.factories.DummyRouter` method), 73
`find_provider_iter()` (`libp2p.routing.interfaces.IContentRouting` method), 53
`FixedSizeMsgReadWriter` (class in `libp2p.io.msgio`), 17
`FloodSub` (class in `libp2p.pubsub.floodsub`), 44
`FloodsubFactory` (class in `libp2p.tools.factories`), 73
`from_base58()` (`libp2p.peer.id.ID` class method), 27
`from_bytes()` (`libp2p.crypto.ecc.ECCPublicKey` class method), 7
`from_bytes()` (`libp2p.crypto.ed25519.Ed25519PrivateKey` class method), 8
`from_bytes()` (`libp2p.crypto.ed25519.Ed25519PublicKey` class method), 8
`from_bytes()` (`libp2p.crypto.rsa.RSAPublicKey` class method), 10
`from_bytes()` (`libp2p.crypto.secp256k1.Secp256k1PrivateKey` class method), 10
`from_bytes()` (`libp2p.crypto.secp256k1.Secp256k1PublicKey` class method), 11
`from_pubkey()` (`libp2p.peer.id.ID` class method), 27
G
`generate_new_rsa_identity()` (in module `libp2p`), 80
`generate_peer_id_from()` (in module `libp2p`), 80
`get()` (`libp2p.peer.peermetadata_interface.IPeerMetadata` method), 31
`get()` (`libp2p.peer.peerstore.PeerStore` method), 32
`get()` (`libp2p.peer.peerstore_interface.IPeerStore` method), 35
`get()` (`libp2p.pubsub.abc.ISubscriptionAPI` method), 43

`get()` (*libp2p.pubsub.mcache.MessageCache* method), 47
`get()` (*libp2p.pubsub.subscription.TrioSubscriptionAPI* method), 52
`get_addrs()` (*libp2p.host.basic_host.BasicHost* method), 12
`get_addrs()` (*libp2p.host.host_interface.IHost* method), 13
`get_addrs()` (*libp2p.peer.peerdata.PeerData* method), 28
`get_addrs()` (*libp2p.peer.peerdata_interface.IPeerData* method), 29
`get_addrs()` (*libp2p.transport.listener_interface.IListener* method), 78
`get_addrs()` (*libp2p.transport.tcp.tcp.TCPListener* method), 77
`get_balance()` (*libp2p.tools.pubsub.dummy_account_node.DummyAccountNode* method), 70
`get_content_addressed_msg_id()` (in module *libp2p.pubsub.pubsub*), 51
`get_default_protocols()` (in module *libp2p.host.defaults*), 13
`get_hello_packet()` (*libp2p.pubsub.pubsub.Pubsub* method), 48
`get_id()` (*libp2p.host.basic_host.BasicHost* method), 12
`get_id()` (*libp2p.host.host_interface.IHost* method), 13
`get_local_peer()` (*libp2p.security.base_session.BaseSession* method), 61
`get_local_peer()` (*libp2p.security.secure_conn_interface.AbstractSecureConn* method), 62
`get_local_private_key()` (*libp2p.security.base_session.BaseSession* method), 61
`get_local_private_key()` (*libp2p.security.secure_conn_interface.AbstractSecureConn* method), 62
`get_metadata()` (*libp2p.peer.peerdata.PeerData* method), 28
`get_metadata()` (*libp2p.peer.peerdata_interface.IPeerData* method), 29
`get_msg_validators()` (*libp2p.pubsub.pubsub.Pubsub* method), 48
`get_mux()` (*libp2p.host.basic_host.BasicHost* method), 12
`get_mux()` (*libp2p.host.host_interface.IHost* method), 13
`get_network()` (*libp2p.host.basic_host.BasicHost* method), 12
`get_network()` (*libp2p.host.host_interface.IHost* method), 14
`get_nonce()` (*libp2p.security.secio.transport.Transport* method), 60
`get_pattern()` (*libp2p.security.noise.transport.Transport* method), 57
`get_peer_and_seqno_msg_id()` (in module *libp2p.pubsub.pubsub*), 51
`get_peer_id()` (*libp2p.network.network_interface.INetwork* method), 22
`get_peer_id()` (*libp2p.network.swarm.Swarm* method), 24
`get_peerstore()` (*libp2p.host.basic_host.BasicHost* method), 12
`get_private_key()` (*libp2p.host.basic_host.BasicHost* method), 12
`get_private_key()` (*libp2p.host.host_interface.IHost* method), 14
`get_privkey()` (*libp2p.peer.peerdata.PeerData* method), 28
`get_privkey()` (*libp2p.peer.peerdata_interface.IPeerData* method), 21
`get_protocol()` (*libp2p.network.stream.net_stream.NetStream* method), 21
`get_protocol()` (*libp2p.network.stream.net_stream_interface.INetStream* method), 21
`get_protocols()` (*libp2p.peer.peerdata.PeerData* method), 28
`get_protocols()` (*libp2p.peer.peerdata_interface.IPeerData* method), 30
`get_protocols()` (*libp2p.peer.peerstore.PeerStore* method), 32
`get_protocols()` (*libp2p.peer.peerstore_interface.IPeerStore* method), 35
`get_protocols()` (*libp2p.protocol_muxer.multiselect_muxer_interface.IMuxer* method), 40
`get_protocols()` (*libp2p.pubsub.abc.IPubsubRouter* method), 42
`get_protocols()` (*libp2p.pubsub.floodsub.FloodSub* method), 44
`get_protocols()` (*libp2p.pubsub.gossipsub.GossipSub* method), 45
`get_pubkey()` (*libp2p.peer.peerdata.PeerData* method), 28
`get_pubkey()` (*libp2p.peer.peerdata_interface.IPeerData* method), 30
`get_public_key()` (*libp2p.crypto.ecc.ECCPrivateKey* method), 7
`get_public_key()` (*libp2p.crypto.ed25519.Ed25519PrivateKey* method), 8
`get_public_key()` (*libp2p.crypto.keys.PrivateKey* method), 9
`get_public_key()` (*libp2p.crypto.rsa.RSAPrivateKey* method), 10
`get_public_key()` (*libp2p.crypto.secp256k1.Secp256k1PrivateKey* method), 10
`get_public_key()` (*libp2p.host.basic_host.BasicHost* method), 12
`get_public_key()` (*libp2p.host.host_interface.IHost* method), 12

method), 14
 get_remote_peer() (libp2p.security.base_session.BaseSession method), 61
 get_remote_peer() (libp2p.security.secure_conn_interface.AbstractSecureConnInterface method), 62
 get_remote_public_key() (libp2p.security.base_session.BaseSession method), 61
 get_remote_public_key() (libp2p.security.secure_conn_interface.AbstractSecureConnInterface method), 62
 get_streams() (libp2p.network.connection.net_connection_interface.INetConnInterface method), 19
 get_streams() (libp2p.network.connection.swarm_connection.SwarmConn method), 20
 get_type() (libp2p.crypto.ecc.ECCPrivateKey method), 7
 get_type() (libp2p.crypto.ecc.ECCPublicKey method), 7
 get_type() (libp2p.crypto.ed25519.Ed25519PrivateKey method), 8
 get_type() (libp2p.crypto.ed25519.Ed25519PublicKey method), 8
 get_type() (libp2p.crypto.keys.Key method), 9
 get_type() (libp2p.crypto.rsa.RSAPrivateKey method), 10
 get_type() (libp2p.crypto.rsa.RSAPublicKey method), 10
 get_type() (libp2p.crypto.secp256k1.Secp256k1PrivateKey method), 10
 get_type() (libp2p.crypto.secp256k1.Secp256k1PublicKey method), 11
 gossip_heartbeat() (libp2p.pubsub.gossipsub.GossipSub method), 45
 gossip_history (libp2p.tools.constants.GossipsubParams attribute), 72
 gossip_history (libp2p.tools.factories.GossipsubFactory attribute), 73
 gossip_window (libp2p.tools.constants.GossipsubParams attribute), 72
 gossip_window (libp2p.tools.factories.GossipsubFactory attribute), 73
 GossipSub (class in libp2p.pubsub.gossipsub), 45
 GossipsubFactory (class in libp2p.tools.factories), 73
 GossipsubParams (class in libp2p.tools.constants), 72
H
 handle_dead_peer_queue() (libp2p.pubsub.pubsub.Pubsub method), 48
 handle_graft() (libp2p.pubsub.gossipsub.GossipSub method), 46
 handle_ihave() (libp2p.pubsub.gossipsub.GossipSub method), 46
 handle_incoming() (libp2p.stream_muxer.mplex.mplex.Mplex method), 65
 handle_incoming_msgs() (libp2p.tools.pubsub.dummy_account_node.DummyAccountNode method), 70
 handle_iwant() (libp2p.pubsub.gossipsub.GossipSub method), 46
 handle_peer_queue() (libp2p.pubsub.pubsub.Pubsub method), 48
 handle_ping() (in module libp2p.host.ping), 14
 handle_prune() (libp2p.pubsub.gossipsub.GossipSub method), 46
 handle_rpc() (libp2p.pubsub.abc.IPubsubRouter method), 43
 handle_rpc() (libp2p.pubsub.floodsub.FloodSub method), 44
 handle_rpc() (libp2p.pubsub.gossipsub.GossipSub method), 46
 handle_send_crypto() (libp2p.tools.pubsub.dummy_account_node.DummyAccountNode method), 70
 handle_set_crypto() (libp2p.tools.pubsub.dummy_account_node.DummyAccountNode method), 71
 handle_subscription() (libp2p.pubsub.pubsub.Pubsub method), 48
 handlers (libp2p.protocol_muxer.multiselect.Multiselect attribute), 37
 handlers (libp2p.protocol_muxer.multiselect_muxer_interface.IMultiselect attribute), 40
 handshake() (libp2p.protocol_muxer.multiselect.Multiselect method), 37
 handshake() (libp2p.protocol_muxer.multiselect_client.MultiselectClient method), 38
 handshake() (libp2p.protocol_muxer.multiselect_client_interface.IMultiselect method), 39
 handshake_inbound() (libp2p.security.noise.patterns.IPattern method), 57
 handshake_inbound() (libp2p.security.noise.patterns.PatternXX method), 57
 handshake_outbound() (libp2p.security.noise.patterns.IPattern method), 57
 handshake_outbound() (libp2p.security.noise.patterns.PatternXX method), 57
 HandshakeFailure, 62
 HandshakeHasNotFinished, 55
 hash_type (libp2p.crypto.authenticated_encryption.EncryptionParameters attribute), 6
 hash_type (libp2p.security.secio.transport.EncryptionParameters

- [attribute](#)), 59
- [hashes](#) (*libp2p.security.secio.transport.Proposal* attribute), 60
- [HeaderTags](#) (class in *libp2p.stream_muxer.mplex.constants*), 64
- [heartbeat\(\)](#) (*libp2p.pubsub.gossipsub.GossipSub* method), 46
- [heartbeat_initial_delay](#) (*libp2p.pubsub.gossipsub.GossipSub* attribute), 46
- [heartbeat_initial_delay](#) (*libp2p.tools.constants.GossipsubParams* attribute), 73
- [heartbeat_initial_delay](#) (*libp2p.tools.factories.GossipsubFactory* attribute), 73
- [heartbeat_interval](#) (*libp2p.pubsub.gossipsub.GossipSub* attribute), 46
- [heartbeat_interval](#) (*libp2p.tools.constants.GossipsubParams* attribute), 73
- [heartbeat_interval](#) (*libp2p.tools.factories.GossipsubFactory* attribute), 73
- [high_watermark](#) (*libp2p.security.secure_session.SecureSession* attribute), 62
- [history](#) (*libp2p.pubsub.mcache.MessageCache* attribute), 47
- [history_size](#) (*libp2p.pubsub.mcache.MessageCache* attribute), 47
- [host](#) (*libp2p.pubsub.pubsub.Pubsub* attribute), 49
- [host](#) (*libp2p.tools.factories.PubsubFactory* attribute), 74
- [host](#) (*libp2p.tools.pubsub.dummy_account_node.DummyAccountNode* property), 71
- [host_pair_factory\(\)](#) (in module *libp2p.tools.factories*), 75
- [HostException](#), 13
- [HostFactory](#) (class in *libp2p.tools.factories*), 73
- |
- [IAddrBook](#) (class in *libp2p.peer.addrbook_interface*), 26
- [IContentRouting](#) (class in *libp2p.routing.interfaces*), 53
- [ID](#) (class in *libp2p.peer.id*), 27
- [id_pubkey](#) (*libp2p.security.noise.messages.NoiseHandshakePayload* attribute), 56
- [id_sig](#) (*libp2p.security.noise.messages.NoiseHandshakePayload* attribute), 56
- [Identify](#) (class in *libp2p.identity.identify.pb.identify_pb2*), 15
- [identify_handler_for\(\)](#) (in module *libp2p.identity.identify.protocol*), 15
- [IdentityHash](#) (class in *libp2p.peer.id*), 27
- [IDFactory](#) (class in *libp2p.tools.factories*), 73
- [IHost](#) (class in *libp2p.host.host_interface*), 13
- [IListener](#) (class in *libp2p.transport.listener_interface*), 78
- [IMultiselectClient](#) (class in *libp2p.protocol_muxer.multiselect_client_interface*), 39
- [IMultiselectCommunicator](#) (class in *libp2p.protocol_muxer.multiselect_communicator_interface*), 40
- [IMultiselectMuxer](#) (class in *libp2p.protocol_muxer.multiselect_muxer_interface*), 40
- [IMuxedConn](#) (class in *libp2p.stream_muxer.abc*), 68
- [IMuxedStream](#) (class in *libp2p.stream_muxer.abc*), 68
- [incoming_data_channel](#) (*libp2p.stream_muxer.mplex.mplex_stream.MplexStream* attribute), 67
- [IncompatibleChoices](#), 59
- [IncompleteReadError](#), 17
- [InconsistentNonce](#), 59
- [INetConn](#) (class in *libp2p.network.connection.net_connection_interface*), 19
- [INetStream](#) (class in *libp2p.network.stream.net_stream_interface*), 21
- [INetwork](#) (class in *libp2p.network.network_interface*), 22
- [INetworkService](#) (class in *libp2p.network.network_interface*), 23
- [infer_local_type\(\)](#) (in module *libp2p.crypto.ecc*), 7
- [info_from_p2p_addr\(\)](#) (in module *libp2p.peer.peerinfo*), 30
- [initialize_pair\(\)](#) (in module *libp2p.crypto.authenticated_encryption*), 6
- [initialize_peerstore_with_our_keypair\(\)](#) (in module *libp2p.tools.factories*), 75
- [initiator_peers_queue](#) (*libp2p.pubsub.pubsub_notiffee.PubsubNotiffee* attribute), 51
- [INotiffee](#) (class in *libp2p.network.notiffee_interface*), 23
- [InsecureSession](#) (class in *libp2p.security.insecure.transport*), 54
- [InsecureTransport](#) (class in *libp2p.security.insecure.transport*), 54
- [InvalidAddrError](#), 30
- [InvalidMACException](#), 6
- [InvalidSignature](#), 55
- [InvalidSignatureOnExchange](#), 59
- [IOException](#), 17
- [IPattern](#) (class in *libp2p.security.noise.patterns*), 57
- [IPeerData](#) (class in *libp2p.peer.peerdata_interface*), 29
- [IPeerMetadata](#) (class in *libp2p.peer.peermetadata_interface*), 31
- [IPeerRouting](#) (class in *libp2p.routing.interfaces*), 53

IPeerStore (class in libp2p.peer.peerstore_interface), 34
 IPubsub (class in libp2p.pubsub.abc), 42
 IPubsubRouter (class in libp2p.pubsub.abc), 42
 IRawConnection (class in module, 80
 libp2p.network.connection.raw_connection_interface), 19
 is_async (libp2p.pubsub.pubsub.TopicValidator attribute), 51
 is_closed (libp2p.network.connection.swarm_connection.SwarmConnection property), 20
 is_closed (libp2p.stream_muxer.abc.IMuxedConn property), 68
 is_closed (libp2p.stream_muxer.mplex.mplex.Mplex property), 65
 is_initiator (libp2p.network.connection.raw_connection.RawConnection attribute), 19
 is_initiator (libp2p.network.connection.raw_connection.RawConnection property), 19
 is_initiator (libp2p.stream_muxer.abc.IMuxedConn property), 68
 is_initiator (libp2p.stream_muxer.mplex.datastructures.StreamData property), 65
 is_initiator (libp2p.stream_muxer.mplex.mplex.Mplex property), 65
 is_initiator (libp2p.stream_muxer.mplex.mplex_stream.MplexStream property), 67
 is_valid_handshake() (in module, 37
 libp2p.protocol_muxer.multiselect), 37
 is_valid_handshake() (in module, 38
 libp2p.protocol_muxer.multiselect_client), 38
 ISecureConn (class in module, 62
 libp2p.security.secure_conn_interface), 62
 ISecureTransport (class in module, 63
 libp2p.security.secure_transport_interface), 63
 ISubscriptionAPI (class in libp2p.pubsub.abc), 43
 ITransport (class in libp2p.transport.transport_interface), 78
 iv (libp2p.crypto.authenticated_encryption.EncryptionParameters attribute), 6

J

join() (libp2p.pubsub.abc.IPubsubRouter method), 43
 join() (libp2p.pubsub.floodsub.FloodSub method), 44
 join() (libp2p.pubsub.gossipsub.GossipSub method), 46

K

Key (class in libp2p.crypto.keys), 9
 KeyPair (class in libp2p.crypto.keys), 9
 KeyType (class in libp2p.crypto.keys), 9

L

leave() (libp2p.pubsub.abc.IPubsubRouter method), 43
 leave() (libp2p.pubsub.floodsub.FloodSub method), 44
 leave() (libp2p.pubsub.gossipsub.GossipSub method), 46
 libp2p
 libp2p.crypto
 libp2p.crypto.authenticated_encryption
 libp2p.crypto.ecc
 libp2p.crypto.ed25519
 libp2p.crypto.exceptions
 libp2p.crypto.key_exchange
 libp2p.crypto.keys
 libp2p.crypto.pb
 libp2p.crypto.pb.crypto_pb2
 libp2p.crypto.rsa
 libp2p.crypto.secp256k1
 libp2p.serialization
 libp2p.exceptions
 libp2p.host
 libp2p.host.basic_host
 libp2p.host.defaults
 libp2p.host.exceptions
 libp2p.host.host_interface
 libp2p.host.ping
 libp2p.host.routed_host
 libp2p.identity
 libp2p.identity.identify
 libp2p.identity.identify.pb
 libp2p.identity.identify.pb.identify_pb2
 libp2p.identity.identify.protocol

- module, 15
- libp2p.io
 - module, 18
- libp2p.io.abc
 - module, 16
- libp2p.io.exceptions
 - module, 17
- libp2p.io.msgio
 - module, 17
- libp2p.io.trio
 - module, 18
- libp2p.io.utils
 - module, 18
- libp2p.network
 - module, 26
- libp2p.network.connection
 - module, 20
- libp2p.network.connection.exceptions
 - module, 18
- libp2p.network.connection.net_connection_interface
 - module, 19
- libp2p.network.connection.raw_connection
 - module, 19
- libp2p.network.connection.raw_connection_interface
 - module, 19
- libp2p.network.connection.swarm_connection
 - module, 20
- libp2p.network.exceptions
 - module, 22
- libp2p.network.network_interface
 - module, 22
- libp2p.network.notifee_interface
 - module, 23
- libp2p.network.stream
 - module, 22
- libp2p.network.stream.exceptions
 - module, 20
- libp2p.network.stream.net_stream
 - module, 21
- libp2p.network.stream.net_stream_interface
 - module, 21
- libp2p.network.swarm
 - module, 24
- libp2p.peer
 - module, 36
- libp2p.peer.addrbook_interface
 - module, 26
- libp2p.peer.id
 - module, 27
- libp2p.peer.peerdata
 - module, 27
- libp2p.peer.peerdata_interface
 - module, 29
- libp2p.peer.peerinfo
 - module, 30
- libp2p.peer.peermetadata_interface
 - module, 31
- libp2p.peer.peerstore
 - module, 31
- libp2p.peer.peerstore_interface
 - module, 34
- libp2p.protocol_muxer
 - module, 41
- libp2p.protocol_muxer.exceptions
 - module, 36
- libp2p.protocol_muxer.multiselect
 - module, 37
- libp2p.protocol_muxer.multiselect_client
 - module, 38
- libp2p.protocol_muxer.multiselect_client_interface
 - module, 39
- libp2p.protocol_muxer.multiselect_communicator
 - module, 39
- libp2p.protocol_muxer.multiselect_communicator_interface
 - module, 40
- libp2p.protocol_muxer.multiselect_muxer_interface
 - module, 40
- libp2p.pubsub
 - module, 53
- libp2p.pubsub.abc
 - module, 42
- libp2p.pubsub.exceptions
 - module, 43
- libp2p.pubsub.floodsub
 - module, 44
- libp2p.pubsub.gossipsub
 - module, 45
- libp2p.pubsub.mcache
 - module, 47
- libp2p.pubsub.pb
 - module, 42
- libp2p.pubsub.pb.rpc_pb2
 - module, 41
- libp2p.pubsub.pubsub
 - module, 48
- libp2p.pubsub.pubsub_notifee
 - module, 51
- libp2p.pubsub.subscription
 - module, 52
- libp2p.pubsub.typing
 - module, 52
- libp2p.pubsub.validators
 - module, 52
- libp2p.routing
 - module, 53
- libp2p.routing.interfaces
 - module, 53
- libp2p.security

module, 64	module, 69
libp2p.security.base_session	libp2p.stream_muxer.mplex
module, 61	module, 68
libp2p.security.base_transport	libp2p.stream_muxer.mplex.constants
module, 62	module, 64
libp2p.security.exceptions	libp2p.stream_muxer.mplex.datastructures
module, 62	module, 65
libp2p.security.insecure	libp2p.stream_muxer.mplex.exceptions
module, 55	module, 65
libp2p.security.insecure.pb	libp2p.stream_muxer.mplex.mplex
module, 54	module, 65
libp2p.security.insecure.pb.plaintext_pb2	libp2p.stream_muxer.mplex.mplex_stream
module, 53	module, 67
libp2p.security.insecure.transport	libp2p.stream_muxer.muxer_multistream
module, 54	module, 69
libp2p.security.noise	libp2p.tools
module, 58	module, 76
libp2p.security.noise.exceptions	libp2p.tools.constants
module, 55	module, 72
libp2p.security.noise.io	libp2p.tools.factories
module, 55	module, 73
libp2p.security.noise.messages	libp2p.tools.pubsub
module, 56	module, 72
libp2p.security.noise.patterns	libp2p.tools.pubsub.dummy_account_node
module, 57	module, 70
libp2p.security.noise.pb	libp2p.tools.pubsub.floodsub_integration_test_settings
module, 55	module, 71
libp2p.security.noise.pb.noise_pb2	libp2p.tools.pubsub.utils
module, 55	module, 72
libp2p.security.noise.transport	libp2p.tools.utils
module, 57	module, 76
libp2p.security.secio	libp2p.transport
module, 61	module, 79
libp2p.security.secio.exceptions	libp2p.transport.exceptions
module, 59	module, 77
libp2p.security.secio.pb	libp2p.transport.listener_interface
module, 59	module, 78
libp2p.security.secio.pb.spice_pb2	libp2p.transport.tcp
module, 58	module, 77
libp2p.security.secio.transport	libp2p.transport.tcp.tcp
module, 59	module, 76
libp2p.security.secure_conn_interface	libp2p.transport.transport_interface
module, 62	module, 78
libp2p.security.secure_session	libp2p.transport.typing
module, 62	module, 79
libp2p.security.secure_transport_interface	libp2p.transport.upgrader
module, 63	module, 79
libp2p.security.security_multistream	libp2p.typing
module, 63	module, 80
libp2p.stream_muxer	libp2p.utils
module, 70	module, 80
libp2p.stream_muxer.abc	libp2p_privkey (<i>libp2p.security.noise.patterns.BasePattern</i>
module, 68	<i>attribute</i>), 57
libp2p.stream_muxer.exceptions	

[libp2p_privkey](#) ([libp2p.security.noise.transport.Transport](#) attribute), 57
[listen\(\)](#) ([libp2p.network.network_interface.INetwork](#) method), 22
[listen\(\)](#) ([libp2p.network.notifee_interface.INotifee](#) method), 23
[listen\(\)](#) ([libp2p.network.swarm.Swarm](#) method), 24
[listen\(\)](#) ([libp2p.pubsub.pubsub_notifee.PubsubNotifee](#) method), 51
[listen\(\)](#) ([libp2p.transport.listener_interface.IListener](#) method), 78
[listen\(\)](#) ([libp2p.transport.tcp.tcp.TCPListener](#) method), 77
[listen_close\(\)](#) ([libp2p.network.notifee_interface.INotifee](#) method), 23
[listen_close\(\)](#) ([libp2p.pubsub.pubsub_notifee.PubsubNotifee](#) method), 51
[listener_nursery](#) ([libp2p.network.swarm.Swarm](#) attribute), 25
[listeners](#) ([libp2p.network.network_interface.INetwork](#) attribute), 22
[listeners](#) ([libp2p.network.swarm.Swarm](#) attribute), 25
[listeners](#) ([libp2p.transport.tcp.tcp.TCPListener](#) attribute), 77
[local_encryption_parameters](#) ([libp2p.security.secio.transport.SessionParameters](#) attribute), 60
[local_peer](#) ([libp2p.security.base_session.BaseSession](#) attribute), 61
[local_peer](#) ([libp2p.security.noise.patterns.BasePattern](#) attribute), 57
[local_peer](#) ([libp2p.security.noise.transport.Transport](#) attribute), 58
[local_peer](#) ([libp2p.security.secio.transport.SessionParameters](#) attribute), 60
[local_private_key](#) ([libp2p.security.base_session.BaseSession](#) attribute), 61
[low_watermark](#) ([libp2p.security.secure_session.SecureSession](#) attribute), 62

M

[mac_key](#) ([libp2p.crypto.authenticated_encryption.EncryptionParameters](#) attribute), 6
[MacAndCipher](#) (class in [libp2p.crypto.authenticated_encryption](#)), 6
[main\(\)](#) (in module [examples.chat.chat](#)), 5
[make_data_to_be_signed\(\)](#) (in module [libp2p.security.noise.messages](#)), 56
[make_exchange_message\(\)](#) (in module [libp2p.security.insecure.transport](#)), 54
[make_handshake_payload\(\)](#) ([libp2p.security.noise.patterns.BasePattern](#) method), 57
[make_handshake_payload_sig\(\)](#) (in module [libp2p.security.noise.messages](#)), 56
[make_pubsub_msg\(\)](#) (in module [libp2p.tools.pubsub.utils](#)), 72
[max_msg_size](#) ([libp2p.io.msgio.VarIntLengthMsgReadWrite](#) attribute), 18
[max_msg_size](#) ([libp2p.security.insecure.transport.PlaintextHandshakeReader](#) attribute), 54
[mcache](#) ([libp2p.pubsub.gossipsub.GossipSub](#) attribute), 46
[mesh](#) ([libp2p.pubsub.gossipsub.GossipSub](#) attribute), 46
[mesh_heartbeat\(\)](#) ([libp2p.pubsub.gossipsub.GossipSub](#) method), 46
[Message](#) (class in [libp2p.pubsub.pb.rpc_pb2](#)), 41
[message_all_peers\(\)](#) ([libp2p.pubsub.pubsub.Pubsub](#) method), 49
[MessageCache](#) (class in [libp2p.pubsub.mcache](#)), 47
[MessageInitiator](#) ([libp2p.stream_muxer.mplex.constants.HeaderTags](#) attribute), 64
[MessageReceiver](#) ([libp2p.stream_muxer.mplex.constants.HeaderTags](#) attribute), 64
[MessageTooLarge](#), 17
[metadata](#) ([libp2p.peer.peerdata.PeerData](#) attribute), 28
[mid](#) ([libp2p.pubsub.mcache.CacheEntry](#) attribute), 47
[MissingDeserializerError](#), 8
[MissingLengthException](#), 17
[MissingMessageException](#), 17
[module](#)
[examples](#), 5
[examples.chat](#), 5
[examples.chat.chat](#), 5
[libp2p](#), 80
[libp2p.crypto](#), 11
[libp2p.crypto.authenticated_encryption](#), 6
[libp2p.crypto.ecc](#), 7
[libp2p.crypto.ed25519](#), 8
[libp2p.crypto.exceptions](#), 8
[libp2p.crypto.key_exchange](#), 8
[libp2p.crypto.keys](#), 9
[libp2p.crypto.pb](#), 6
[libp2p.crypto.pb.crypto_pb2](#), 6
[libp2p.crypto.rsa](#), 10
[libp2p.crypto.secp256k1](#), 10
[libp2p.crypto.serialization](#), 11
[libp2p.exceptions](#), 79
[libp2p.host](#), 15
[libp2p.host.basic_host](#), 11
[libp2p.host.defaults](#), 13
[libp2p.host.exceptions](#), 13
[libp2p.host.host_interface](#), 13
[libp2p.host.ping](#), 14
[libp2p.host.routed_host](#), 15
[libp2p.identity](#), 16
[libp2p.identity.identify](#), 16

libp2p.identity.identify.pb, 15
 libp2p.identity.identify.pb.identify_pb2, 15
 libp2p.identity.identify.protocol, 15
 libp2p.io, 18
 libp2p.io.abc, 16
 libp2p.io.exceptions, 17
 libp2p.io.msgio, 17
 libp2p.io.trio, 18
 libp2p.io.utils, 18
 libp2p.network, 26
 libp2p.network.connection, 20
 libp2p.network.connection.exceptions, 18
 libp2p.network.connection.net_connection_interface, 19
 libp2p.network.connection.raw_connection, 19
 libp2p.network.connection.raw_connection_interface, 19
 libp2p.network.connection.swarm_connection, 20
 libp2p.network.exceptions, 22
 libp2p.network.network_interface, 22
 libp2p.network.notifee_interface, 23
 libp2p.network.stream, 22
 libp2p.network.stream.exceptions, 20
 libp2p.network.stream.net_stream, 21
 libp2p.network.stream.net_stream_interface, 21
 libp2p.network.swarm, 24
 libp2p.peer, 36
 libp2p.peer.addrbook_interface, 26
 libp2p.peer.id, 27
 libp2p.peer.peerdata, 27
 libp2p.peer.peerdata_interface, 29
 libp2p.peer.peerinfo, 30
 libp2p.peer.peermetadata_interface, 31
 libp2p.peer.peerstore, 31
 libp2p.peer.peerstore_interface, 34
 libp2p.protocol_muxer, 41
 libp2p.protocol_muxer.exceptions, 36
 libp2p.protocol_muxer.multiselect, 37
 libp2p.protocol_muxer.multiselect_client, 38
 libp2p.protocol_muxer.multiselect_client_interface, 39
 libp2p.protocol_muxer.multiselect_communicator, 39
 libp2p.protocol_muxer.multiselect_communicator_interface, 40
 libp2p.protocol_muxer.multiselect_muxer_interface, 40
 libp2p.pubsub, 53
 libp2p.pubsub.abc, 42
 libp2p.pubsub.exceptions, 43
 libp2p.pubsub.floodsub, 44
 libp2p.pubsub.gossipsub, 45
 libp2p.pubsub.mcache, 47
 libp2p.pubsub.pb, 42
 libp2p.pubsub.pb.rpc_pb2, 41
 libp2p.pubsub.pubsub, 48
 libp2p.pubsub.pubsub_notifee, 51
 libp2p.pubsub.subscription, 52
 libp2p.pubsub.typing, 52
 libp2p.pubsub.validators, 52
 libp2p.routing, 53
 libp2p.routing.interfaces, 53
 libp2p.security, 64
 libp2p.security.base_session, 61
 libp2p.security.base_transport, 62
 libp2p.security.exceptions, 62
 libp2p.security.insecure, 55
 libp2p.security.insecure.pb, 54
 libp2p.security.insecure.pb.plaintext_pb2, 53
 libp2p.security.insecure.transport, 54
 libp2p.security.noise, 58
 libp2p.security.noise.exceptions, 55
 libp2p.security.noise.io, 55
 libp2p.security.noise.messages, 56
 libp2p.security.noise.patterns, 57
 libp2p.security.noise.pb, 55
 libp2p.security.noise.pb.noise_pb2, 55
 libp2p.security.noise.transport, 57
 libp2p.security.secio, 61
 libp2p.security.secio.exceptions, 59
 libp2p.security.secio.pb, 59
 libp2p.security.secio.pb.spice_pb2, 58
 libp2p.security.secio.transport, 59
 libp2p.security.secure_conn_interface, 62
 libp2p.security.secure_session, 62
 libp2p.security.secure_transport_interface, 63
 libp2p.security.security_multistream, 63
 libp2p.stream_muxer, 70
 libp2p.stream_muxer.abc, 68
 libp2p.stream_muxer.exceptions, 69
 libp2p.stream_muxer.mplex, 68
 libp2p.stream_muxer.mplex.constants, 64
 libp2p.stream_muxer.mplex.datastructures, 65
 libp2p.stream_muxer.mplex.exceptions, 65
 libp2p.stream_muxer.mplex.mplex, 65
 libp2p.stream_muxer.mplex.mplex_stream, 65
 libp2p.stream_muxer.muxer_multistream, 69
 libp2p.tools, 76
 libp2p.tools.constants, 72

libp2p.tools.factories, 73
 libp2p.tools.pubsub, 72
 libp2p.tools.pubsub.dummy_account_node, 70
 libp2p.tools.pubsub.floodsub_integration_tests, 71
 libp2p.tools.pubsub.utils, 72
 libp2p.tools.utils, 76
 libp2p.transport, 79
 libp2p.transport.exceptions, 77
 libp2p.transport.listener_interface, 78
 libp2p.transport.tcp, 77
 libp2p.transport.tcp.tcp, 76
 libp2p.transport.transport_interface, 78
 libp2p.transport.typing, 79
 libp2p.transport.upgrader, 79
 libp2p.typing, 80
 libp2p.utils, 80
 Mplex (class in libp2p.stream_muxer.mplex.mplex), 65
 mplex_conn_pair_factory() (in module libp2p.tools.factories), 75
 mplex_stream_pair_factory() (in module libp2p.tools.factories), 75
 mplex_transport_factory() (in module libp2p.tools.factories), 75
 MplexError, 65
 MplexStream (class in libp2p.stream_muxer.mplex.mplex_stream), 67
 MplexStreamClosed, 65
 MplexStreamEOF, 65
 MplexStreamReset, 65
 MplexUnavailable, 65
 MsgioException, 17
 MsgReader (class in libp2p.io.abc), 16
 MsgReadWriteCloser (class in libp2p.io.abc), 16
 msgs (libp2p.pubsub.mcache.MessageCache attribute), 47
 MsgWriter (class in libp2p.io.abc), 16
 MultiError, 79
 Multiselect (class in libp2p.protocol_muxer.multiselect), 37
 multiselect (libp2p.host.basic_host.BasicHost attribute), 12
 multiselect (libp2p.security.security_multistream.SecurityMultistream attribute), 63
 multiselect (libp2p.stream_muxer.muxer_multistream.MuxerMultistream attribute), 69
 multiselect_client (libp2p.host.basic_host.BasicHost attribute), 12
 multiselect_client (libp2p.security.security_multistream.SecurityMultistream attribute), 63
 multiselect_client (libp2p.stream_muxer.muxer_multistream.MuxerMultistream attribute), 69
 MultiselectClient (class in libp2p.protocol_muxer.multiselect_client), 38
 MultiselectClientError, 36
 MultiselectCommunicator (class in libp2p.protocol_muxer.multiselect_communicator), 39
 MultiselectCommunicatorError, 36
 MultiselectError, 36
 muxed_conn (libp2p.network.connection.net_connection_interface.INetConnection attribute), 19
 muxed_conn (libp2p.network.connection.swarm_connection.SwarmConn attribute), 20
 muxed_conn (libp2p.network.stream.net_stream_interface.INetStream attribute), 21
 muxed_conn (libp2p.stream_muxer.abc.IMuxedStream attribute), 68
 muxed_conn (libp2p.stream_muxer.mplex.mplex_stream.MplexStream attribute), 67
 muxed_stream (libp2p.network.stream.net_stream.NetStream attribute), 21
 MuxedConnError, 69
 MuxedConnUnavailable, 69
 MuxedStreamClosed, 69
 MuxedStreamEOF, 69
 MuxedStreamError, 69
 MuxedStreamReset, 69
 muxer_multistream (libp2p.transport.upgrader.TransportUpgrader attribute), 79
 MuxerMultistream (class in libp2p.stream_muxer.muxer_multistream), 69
 MuxerUpgradeFailure, 77
 my_id (libp2p.pubsub.abc.IPubsub property), 42
 my_id (libp2p.pubsub.pubsub.Pubsub property), 49
N
 name (libp2p.stream_muxer.mplex.mplex_stream.MplexStream attribute), 67
 negotiate() (libp2p.protocol_muxer.multiselect.Multiselect method), 37
 negotiate() (libp2p.protocol_muxer.multiselect_muxer_interface.IMultiselect method), 40
 net_stream_pair_factory() (in module libp2p.tools.factories), 75
 NetStream (class in libp2p.network.stream.net_stream), 21
 network (libp2p.tools.factories.HostFactory attribute), 73
 network (libp2p.tools.factories.RoutedHostFactory attribute), 73
 new() (libp2p.crypto.ecc.ECCPrivateKey class method), 42

[new\(\)](#) (*libp2p.crypto.ed25519.Ed25519PrivateKey* class method), 8
[new\(\)](#) (*libp2p.crypto.rsa.RSAPrivateKey* class method), 10
[new\(\)](#) (*libp2p.crypto.secp256k1.Secp256k1PrivateKey* class method), 10
[new_conn\(\)](#) (*libp2p.stream_muxer.muxer_multistream.MuxerMultistreamer* class method), 69
[new_host\(\)](#) (in module *libp2p*), 80
[new_stream\(\)](#) (*libp2p.host.basic_host.BasicHost* class method), 12
[new_stream\(\)](#) (*libp2p.host.host_interface.IHost* class method), 14
[new_stream\(\)](#) (*libp2p.network.connection.net_connection.NetConnection* class method), 19
[new_stream\(\)](#) (*libp2p.network.connection.swarm_connection.SwarmConnection* class method), 20
[new_stream\(\)](#) (*libp2p.network.network_interface.INetworkInterface* class method), 22
[new_stream\(\)](#) (*libp2p.network.swarm.Swarm* method), 25
[new_stream_receive_channel](#) (*libp2p.stream_muxer.mplex.mplex.Mplex* attribute), 66
[new_stream_send_channel](#) (*libp2p.stream_muxer.mplex.mplex.Mplex* attribute), 66
[new_swarm\(\)](#) (in module *libp2p*), 80
[NewStream](#) (*libp2p.stream_muxer.mplex.constants.HeaderTags* attribute), 64
[next_channel_id](#) (*libp2p.stream_muxer.mplex.mplex.Mplex* attribute), 66
[next_msg_len\(\)](#) (*libp2p.io.msgio.BaseMsgReadWriter* class method), 17
[next_msg_len\(\)](#) (*libp2p.io.msgio.FixedSizeLenMsgReadWriter* class method), 18
[next_msg_len\(\)](#) (*libp2p.io.msgio.VarIntLengthMsgReadWriter* class method), 18
[noise_conn_factory\(\)](#) (in module *libp2p.tools.factories*), 75
[noise_handshake_payload_factory\(\)](#) (in module *libp2p.tools.factories*), 75
[noise_privkey](#) (*libp2p.security.noise.transport.Transport* attribute), 58
[noise_state](#) (*libp2p.security.noise.io.BaseNoiseMsgReadWriter* attribute), 56
[noise_static_key](#) (*libp2p.security.noise.patterns.BasePattern* attribute), 57
[noise_static_key_factory\(\)](#) (in module *libp2p.tools.factories*), 75
[noise_transport_factory\(\)](#) (in module *libp2p.tools.factories*), 75
[NoiseFailure](#), 55
[NoiseHandshakePayload](#) (class in *libp2p.security.noise.messages*), 56
[NoiseHandshakePayload](#) (class in *libp2p.security.noise.pb.noise_pb2*), 55
[NoiseHandshakeReadWriter](#) (class in *libp2p.security.noise.io*), 56
[NoisePacketReadWriter](#) (class in *libp2p.security.noise.io*), 56
[NoiseStateError](#), 55
[NoiseTransportReadWriter](#) (class in *libp2p.security.noise.io*), 56
[nonce](#) (*libp2p.security.secio.transport.Proposal* attribute), 60
[NoPubsubAttached](#), 43
[notify_closed_stream\(\)](#) (*libp2p.network.swarm.Swarm* attribute), 25
[notify_connected\(\)](#) (*libp2p.network.swarm.Swarm* method), 25
[notify_disconnected\(\)](#) (*libp2p.network.swarm.Swarm* method), 25
[notify_listen\(\)](#) (*libp2p.network.swarm.Swarm* method), 25
[notify_listen_close\(\)](#) (*libp2p.network.swarm.Swarm* method), 25
[notify_opened_stream\(\)](#) (*libp2p.network.swarm.Swarm* method), 25
[notify_subscriptions\(\)](#) (*libp2p.pubsub.pubsub.Pubsub* method), 49
[one_to_all_connect\(\)](#) (in module *libp2p.tools.pubsub.utils*), 72
[open_stream\(\)](#) (*libp2p.stream_muxer.abc.IMuxedConn* method), 68
[open_stream\(\)](#) (*libp2p.stream_muxer.mplex.mplex.Mplex* method), 66
[OpenConnectionError](#), 77
[opened_stream\(\)](#) (*libp2p.network.notifee_interface.INotifee* method), 23
[opened_stream\(\)](#) (*libp2p.pubsub.pubsub_notifee.PubsubNotifee* method), 52
[Order](#) (*libp2p.security.secio.transport.SessionParameters* attribute), 60

P

[pack_control_msgs\(\)](#) (*libp2p.pubsub.gossipsub.GossipSub* method), 46
[ParseError](#), 79

PatternXX (class in libp2p.security.noise.patterns), 57
 peer_data_map (libp2p.peer.peerstore.PeerStore attribute), 33
 peer_id (libp2p.peer.peerinfo.PeerInfo attribute), 30
 peer_id (libp2p.stream_muxer.abc.IMuxedConn attribute), 68
 peer_id (libp2p.stream_muxer.mplex.mplex.Mplex attribute), 66
 peer_id (libp2p.tools.factories.SwarmFactory attribute), 75
 peer_id_bytes (libp2p.tools.factories.IDFactory attribute), 74
 peer_ids() (libp2p.peer.peerstore.PeerStore method), 33
 peer_ids() (libp2p.peer.peerstore_interface.IPeerStore method), 35
 peer_info() (libp2p.peer.peerstore.PeerStore method), 33
 peer_info() (libp2p.peer.peerstore_interface.IPeerStore method), 35
 peer_protocol (libp2p.pubsub.gossipsub.GossipSub attribute), 46
 peer_receive_channel (libp2p.pubsub.pubsub.Pubsub attribute), 49
 peer_topics (libp2p.pubsub.pubsub.Pubsub attribute), 49
 PeerData (class in libp2p.peer.peerdata), 27
 PeerDataError, 29
 PeerIDMismatchesPubkey, 55
 PeerInfo (class in libp2p.peer.peerinfo), 30
 PeerMismatchException, 59
 peers (libp2p.pubsub.pubsub.Pubsub attribute), 49
 peers_with_addrs() (libp2p.peer.addrbook_interface.IAddrbook method), 27
 peers_with_addrs() (libp2p.peer.peerstore.PeerStore method), 33
 peers_with_addrs() (libp2p.peer.peerstore_interface.IPeerStore method), 35
 PeerStore (class in libp2p.peer.peerstore), 31
 peerstore (libp2p.host.basic_host.BasicHost attribute), 12
 peerstore (libp2p.network.network_interface.INetwork attribute), 22
 peerstore (libp2p.network.swarm.Swarm attribute), 25
 peerstore (libp2p.tools.factories.SwarmFactory attribute), 75
 PeerStoreError, 33
 perform_test_from_obj() (in module libp2p.tools.pubsub.floodsub_integration_test_settings), 71
 permanent_public_key (libp2p.security.secio.transport.EncryptionParameters attribute), 59
 plaintext_transport_factory() (in module libp2p.tools.factories), 75
 PlaintextHandshakeReadWriter (class in libp2p.security.insecure.transport), 54
 prefix (libp2p.security.noise.io.BaseNoiseMsgReadWriter attribute), 56
 pretty() (libp2p.peer.id.ID method), 27
 private_key (libp2p.crypto.keys.KeyPair attribute), 9
 PrivateKey (class in libp2p.crypto.keys), 9
 PrivateKey (class in libp2p.crypto.pb.crypto_pb2), 6
 privkey (libp2p.peer.peerdata.PeerData attribute), 28
 privkey() (libp2p.peer.peerstore.PeerStore method), 33
 privkey() (libp2p.peer.peerstore_interface.IPeerStore method), 35
 Proposal (class in libp2p.security.secio.transport), 59
 Propose (class in libp2p.security.secio.pb.spiffe_pb2), 58
 protocol_id (libp2p.network.stream.net_stream.NetStream attribute), 21
 protocol_name (libp2p.security.noise.patterns.BasePattern attribute), 57
 protocols (libp2p.peer.peerdata.PeerData attribute), 28
 protocols (libp2p.pubsub.abc.IPubsub property), 42
 protocols (libp2p.pubsub.floodsub.FloodSub attribute), 44
 protocols (libp2p.pubsub.gossipsub.GossipSub attribute), 46
 protocols (libp2p.pubsub.pubsub.Pubsub property), 49
 protocols (libp2p.tools.factories.FloodsubFactory attribute), 73
 protocols (libp2p.tools.factories.GossipsubFactory attribute), 73
 provide() (libp2p.routing.interfaces.IContentRouting method), 53
 pubkey (libp2p.peer.peerdata.PeerData attribute), 28
 pubkey() (libp2p.peer.peerstore.PeerStore method), 33
 pubkey() (libp2p.peer.peerstore_interface.IPeerStore method), 36
 public_key (libp2p.crypto.keys.KeyPair attribute), 9
 public_key (libp2p.security.secio.transport.Proposal attribute), 60
 PublicKey (class in libp2p.crypto.keys), 9
 PublicKey (class in libp2p.crypto.pb.crypto_pb2), 6
 publish() (libp2p.pubsub.abc.IPubsub method), 42
 publish() (libp2p.pubsub.abc.IPubsubRouter method), 43
 publish() (libp2p.pubsub.floodsub.FloodSub method), 44
 publish() (libp2p.pubsub.gossipsub.GossipSub method), 46
 publish() (libp2p.pubsub.pubsub.Pubsub method), 49
 publish_send_crypto() (libp2p.tools.pubsub.dummy_account_node.DummyAccountNode method), 71
 publish_set_crypto()

(*libp2p.tools.pubsub.dummy_account_node.DummyAccountNode* attribute), 18
read_lock() (*libp2p.io.trio.TrioTCPStream* attribute), 18
read_message() (*libp2p.stream_muxer.mplex.mplex.Mplex* method), 66
read_msg() (*libp2p.io.abc.MsgReader* method), 16
read_msg() (*libp2p.io.msgio.BaseMsgReadWriter* method), 17
read_msg() (*libp2p.security.noise.io.BaseNoiseMsgReadWriter* method), 56
read_msg() (*libp2p.security.secio.transport.SecioMsgReadWriter* method), 60
read_varint_prefixed_bytes() (in module *libp2p.utils*), 80
read_write_closer() (*libp2p.io.msgio.BaseMsgReadWriter* attribute), 17
read_writer() (*libp2p.protocol_muxer.multiselect_communicator.MultiselectCommunicator* attribute), 39
read_writer() (*libp2p.security.noise.io.BaseNoiseMsgReadWriter* attribute), 56
read_writer() (*libp2p.security.secio.transport.SecioMsgReadWriter* attribute), 60
ReadCloser (class in *libp2p.io.abc*), 16
Reader (class in *libp2p.io.abc*), 16
ReadWriteCloser (class in *libp2p.io.abc*), 16
ReadWriter (class in *libp2p.io.abc*), 16
receive_channel() (*libp2p.pubsub.subscription.TrioSubscriptionAPI* attribute), 52
register_notifiee() (*libp2p.network.network_interface.INetworkInterface* method), 23
register_notifiee() (*libp2p.network.swarm.Swarm* method), 25
remote_encryption_parameters (*libp2p.security.secio.transport.SessionParameters* attribute), 60
remote_peer() (*libp2p.security.base_session.BaseSession* attribute), 61
remote_peer() (*libp2p.security.secio.transport.SessionParameters* attribute), 60
remote_permanent_pubkey (*libp2p.security.base_session.BaseSession* attribute), 61
remove_conn() (*libp2p.network.swarm.Swarm* method), 25
remove_conn() (*libp2p.protocol_muxer.multiselect_communicator.MultiselectCommunicator* method), 40
remove_peer() (*libp2p.pubsub.abc.IPubsubRouter* method), 43
remove_peer() (*libp2p.pubsub.floodsub.FloodSub* method), 44
remove_peer() (*libp2p.pubsub.gossipsub.GossipSub* method), 47
remove_stream() (*libp2p.network.connection.swarm_connection.SwarmConnection* method), 20
remove_topic_validator() (*libp2p.pubsub.abc.IPubsub* method), 42
remove_topic_validator() (*libp2p.pubsub.pubsub.Pubsub* method), 48
Pubsub (class in *libp2p.pubsub.pubsub*), 48
pubsub (*libp2p.pubsub.floodsub.FloodSub* attribute), 44
pubsub (*libp2p.pubsub.gossipsub.GossipSub* attribute), 46
pubsub (*libp2p.tools.pubsub.dummy_account_node.DummyAccountNode* attribute), 71
PubsubFactory (class in *libp2p.tools.factories*), 74
PubsubNotifiee (class in *libp2p.pubsub.pubsub_notifiee*), 51
PubsubRouterError, 43
push_msg() (*libp2p.pubsub.pubsub.Pubsub* method), 49
put() (*libp2p.peer.peermetadata_interface.IPeerMetadata* method), 31
put() (*libp2p.peer.peerstore.PeerStore* method), 33
put() (*libp2p.peer.peerstore_interface.IPeerStore* method), 36
put() (*libp2p.pubsub.mcache.MessageCache* method), 47
put_metadata() (*libp2p.peer.peerdata.PeerData* method), 29
put_metadata() (*libp2p.peer.peerdata_interface.IPeerDataInterface* method), 30

R

raw_conn_factory() (in module *libp2p.tools.factories*), 75
RawConnection (class in *libp2p.network.connection.raw_connection*), 19
RawConnError, 18
read() (*libp2p.io.abc.Reader* method), 16
read() (*libp2p.io.trio.TrioTCPStream* method), 18
read() (*libp2p.network.connection.raw_connection.RawConnection* method), 19
read() (*libp2p.network.stream.net_stream.NetStream* method), 21
read() (*libp2p.protocol_muxer.multiselect_communicator.MultiselectCommunicator* method), 39
read() (*libp2p.protocol_muxer.multiselect_communicator.MultiselectCommunicator* method), 40
read() (*libp2p.security.insecure.transport.InsecureSession* method), 54
read() (*libp2p.security.secure_session.SecureSession* method), 63
read() (*libp2p.stream_muxer.mplex.mplex_stream.MplexStream* method), 67
read_data() (in module *examples.chat.chat*), 5
read_deadline() (*libp2p.stream_muxer.mplex.mplex_stream.MplexStream* attribute), 67
read_delim() (in module *libp2p.utils*), 80
read_exactly() (in module *libp2p.io.utils*), 18
read_length() (in module *libp2p.io.msgio*), 18

49
 reset() (*libp2p.network.stream.net_stream.NetStream* method), 21
 reset() (*libp2p.network.stream.net_stream_interface.INetStream* method), 21
 reset() (*libp2p.stream_muxer.abc.IMuxedStream* method), 69
 reset() (*libp2p.stream_muxer.mplex.mplex_stream.MplexStream* method), 67
 ResetInitiator (*libp2p.stream_muxer.mplex.constants.HeaderTags* method), 58
 attribute), 64
 ResetReceiver (*libp2p.stream_muxer.mplex.constants.HeaderTags* method), 61
 attribute), 64
 RoutedHost (class in *libp2p.host.routed_host*), 15
 RoutedHostFactory (class in *libp2p.tools.factories*), 74
 router (*libp2p.pubsub.pubsub.Pubsub* attribute), 49
 router (*libp2p.tools.factories.PubsubFactory* attribute), 74
 router (*libp2p.tools.factories.RoutedHostFactory* attribute), 74
 RPC (class in *libp2p.pubsub.pb.rpc_pb2*), 41
 RPC.SubOpts (class in *libp2p.pubsub.pb.rpc_pb2*), 41
 RSA (*libp2p.crypto.keys.KeyType* attribute), 9
 RSAPrivateKey (class in *libp2p.crypto.rsa*), 10
 RSAPublicKey (class in *libp2p.crypto.rsa*), 10
 run() (in module *examples.chat.chat*), 5
 run() (*libp2p.host.basic_host.BasicHost* method), 12
 run() (*libp2p.host.host_interface.IHost* method), 14
 run() (*libp2p.network.swarm.Swarm* method), 26
 run() (*libp2p.pubsub.gossipsub.GossipSub* method), 47
 run() (*libp2p.pubsub.pubsub.Pubsub* method), 49
 run() (*libp2p.tools.pubsub.dummy_account_node.DummyAccountNode* method), 71
 run_handshake() (in module *libp2p.security.insecure.transport*), 54

S

secio_transport_factory() (in module *libp2p.tools.factories*), 75
 SecioException, 59
 SecioMsgReadWriter (class in *libp2p.security.secio.transport*), 60
 SecioPacketReadWriter (class in *libp2p.security.secio.transport*), 60
 Secp256k1 (*libp2p.crypto.keys.KeyType* attribute), 9
 Secp256k1PrivateKey (class in *libp2p.crypto.secp256k1*), 10
 Secp256k1PublicKey (class in *libp2p.crypto.secp256k1*), 10
 secure_inbound() (*libp2p.security.insecure.transport.InsecureTransport* method), 54
 secure_inbound() (*libp2p.security.noise.transport.Transport* method), 58
 secure_inbound() (*libp2p.security.secio.transport.Transport* method), 60
 secure_inbound() (*libp2p.security.secure_transport_interface.ISecureTransport* method), 63
 secure_inbound() (*libp2p.security.security_multistream.SecurityMultistream* method), 63
 secure_outbound() (*libp2p.security.insecure.transport.InsecureTransport* method), 54
 secure_outbound() (*libp2p.security.noise.transport.Transport* method), 63
 secure_outbound() (*libp2p.security.secio.transport.Transport* method), 61
 secure_outbound() (*libp2p.security.secure_transport_interface.ISecureTransport* method), 63
 secure_outbound() (*libp2p.security.security_multistream.SecurityMultistream* method), 63
 secured_conn (*libp2p.stream_muxer.mplex.mplex.Mplex* attribute), 66
 SecureSession (class in *libp2p.security.secure_session*), 62
 security_multistream (*libp2p.transport.upgrader.TransportUpgrader* attribute), 79
 security_options_factory_factory() (in module *libp2p.tools.factories*), 75
 SecurityMultistream (class in *libp2p.security.security_multistream*), 63
 SecurityUpgradeFailure, 77
 SedesException, 59
 seen_messages (*libp2p.pubsub.pubsub.Pubsub* attribute), 50
 select_one_of() (*libp2p.pubsub.gossipsub.GossipSub* static method), 47
 select_one_of() (*libp2p.protocol_muxer.multiselect_client.MultiselectClient* method), 38
 select_one_of() (*libp2p.protocol_muxer.multiselect_client_interface.IMultiselectClient* method), 39
 select_transport() (*libp2p.security.security_multistream.SecurityMultistream* method), 64
 select_transport() (*libp2p.stream_muxer_muxer_multistream.MuxerMultistream* method), 70
 self_id (*libp2p.network.swarm.Swarm* attribute), 26
 SelfEncryption, 59
 send_message() (*libp2p.stream_muxer.mplex.mplex.Mplex* method), 66
 serialize() (*libp2p.crypto.keys.PrivateKey* method), 9
 serialize() (*libp2p.crypto.keys.PublicKey* method), 9
 serialize() (*libp2p.security.noise.messages.NoiseHandshakePayload* method), 56
 serialize() (*libp2p.security.secio.transport.Proposal* method), 60
 SessionParameters (class in *libp2p.security.secio.transport*), 60

[set_deadline\(\)](#) ([libp2p.stream_muxer.abc.IMuxedStream](#)[size_len_bytes](#) ([libp2p.io.msgio.FixedSizeLenMsgReadWriter](#) attribute), 69
[set_deadline\(\)](#) ([libp2p.stream_muxer.mplex.mplex_stream.MplexStream](#)[size_len_bytes](#) ([libp2p.security.noise.io.NoisePacketReadWriter](#) attribute), 67
[set_protocol\(\)](#) ([libp2p.network.stream.net_stream.NetStream](#)[size_len_bytes](#) ([libp2p.security.secio.transport.SecioPacketReadWriter](#) attribute), 21
[set_protocol\(\)](#) ([libp2p.network.stream.net_stream_interface.INetStream](#) ([libp2p.network.connection.swarm_connection.SwarmConn](#) method), 21
[set_protocols\(\)](#) ([libp2p.peer.peerdata.PeerData](#) [start\(\)](#) ([libp2p.stream_muxer.abc.IMuxedConn](#) method), 29
[set_protocols\(\)](#) ([libp2p.peer.peerdata_interface.IPeerData](#) [start\(\)](#) ([libp2p.stream_muxer.mplex.mplex.Mplex](#) method), 30
[set_protocols\(\)](#) ([libp2p.peer.peerstore.PeerStore](#) [stream](#) ([libp2p.io.trio.TrioTCPStream](#) attribute), 33
[set_protocols\(\)](#) ([libp2p.peer.peerstore_interface.IPeerStore](#) [stream](#) ([libp2p.network.connection.raw_connection.RawConnection](#) attribute), 36
[set_read_deadline\(\)](#) ([libp2p.stream_muxer.mplex.mplex_stream.MplexStream](#)[stream_id](#) ([libp2p.stream_muxer.mplex.mplex_stream.MplexStream](#) attribute), 67
[set_stream_handler\(\)](#) ([libp2p.host.basic_host.BasicHost](#) method), 12
[set_stream_handler\(\)](#) ([libp2p.host.host_interface.IHost](#) method), 14
[set_stream_handler\(\)](#) ([libp2p.network.network_interface.INetwork](#) method), 23
[set_stream_handler\(\)](#) ([libp2p.network.swarm.Swarm](#) method), 26
[set_topic_validator\(\)](#) ([libp2p.pubsub.abc.IPubsub](#) method), 42
[set_topic_validator\(\)](#) ([libp2p.pubsub.pubsub.Pubsub](#) method), 50
[set_write_deadline\(\)](#) ([libp2p.stream_muxer.mplex.mplex_stream.MplexStream](#)[strict_signing](#) ([libp2p.pubsub.pubsub.Pubsub](#) attribute), 67
[sha256_digest\(\)](#) (in module [libp2p.peer.id](#)), 27
[shared_key](#) ([libp2p.security.secio.transport.SessionParameters](#) attribute), 60
[shift\(\)](#) ([libp2p.pubsub.mcache.MessageCache](#) method), 47
[sign\(\)](#) ([libp2p.crypto.ecc.ECCPrivateKey](#) method), 7
[sign\(\)](#) ([libp2p.crypto.ed25519.Ed25519PrivateKey](#) method), 8
[sign\(\)](#) ([libp2p.crypto.keys.PrivateKey](#) method), 9
[sign\(\)](#) ([libp2p.crypto.rsa.RSAPrivateKey](#) method), 10
[sign\(\)](#) ([libp2p.crypto.secp256k1.Secp256k1PrivateKey](#) method), 10
[sign_key](#) ([libp2p.pubsub.pubsub.Pubsub](#) attribute), 50
[signature_validator\(\)](#) (in module [libp2p.pubsub.validators](#)), 52
[size_len_bytes](#) ([libp2p.io.msgio.BaseMsgReadWriter](#) attribute), 17
[StreamClosed](#), 20
[StreamEOF](#), 20
[StreamError](#), 20
[StreamFailure](#), 13
[StreamID](#) (class in [libp2p.stream_muxer.mplex.datastructures](#)), 65
[StreamReset](#), 20
[streams](#) ([libp2p.network.connection.swarm_connection.SwarmConn](#) attribute), 20
[streams](#) ([libp2p.stream_muxer.mplex.mplex.Mplex](#) attribute), 66
[streams_lock](#) ([libp2p.stream_muxer.mplex.mplex.Mplex](#) attribute), 66
[streams_msg_channels](#) ([libp2p.stream_muxer.mplex.mplex.Mplex](#) attribute), 66
[strict_signing](#) ([libp2p.pubsub.pubsub.Pubsub](#) attribute), 50
[strict_signing](#) ([libp2p.tools.factories.PubsubFactory](#) attribute), 74
[subscribe\(\)](#) ([libp2p.pubsub.abc.IPubsub](#) method), 42
[subscribe\(\)](#) ([libp2p.pubsub.pubsub.Pubsub](#) method), 50
[subscribed_topics_receive](#) ([libp2p.pubsub.pubsub.Pubsub](#) attribute), 50
[subscribed_topics_send](#) ([libp2p.pubsub.pubsub.Pubsub](#) attribute), 50
[Swarm](#) (class in [libp2p.network.swarm](#)), 24
[swarm](#) ([libp2p.network.connection.swarm_connection.SwarmConn](#) attribute), 20
[swarm_conn_pair_factory\(\)](#) (in module [libp2p.tools.factories](#)), 76
[swarm_pair_factory\(\)](#) (in module

libp2p.tools.factories), 76
 SwarmConn (class in *libp2p.network.connection.swarm_connection*), attribute), 64
 20
 SwarmException, 22
 SwarmFactory (class in *libp2p.tools.factories*), 74

T

TCP (class in *libp2p.transport.tcp.tcp*), 76
 TCPListener (class in *libp2p.transport.tcp.tcp*), 77
 time_to_live (*libp2p.pubsub.gossipsub.GossipSub* attribute), 47
 time_to_live (*libp2p.tools.constants.GossipsubParams* attribute), 73
 time_to_live (*libp2p.tools.factories.GossipsubFactory* attribute), 73
 to_base58() (*libp2p.peer.id.ID* method), 27
 to_bytes() (*libp2p.crypto.ecc.ECCPrivateKey* method), 7
 to_bytes() (*libp2p.crypto.ecc.ECCPublicKey* method), 7
 to_bytes() (*libp2p.crypto.ed25519.Ed25519PrivateKey* method), 8
 to_bytes() (*libp2p.crypto.ed25519.Ed25519PublicKey* method), 8
 to_bytes() (*libp2p.crypto.keys.Key* method), 9
 to_bytes() (*libp2p.crypto.rsa.RSAPrivateKey* method), 10
 to_bytes() (*libp2p.crypto.rsa.RSAPublicKey* method), 10
 to_bytes() (*libp2p.crypto.secp256k1.Secp256k1PrivateKey* method), 10
 to_bytes() (*libp2p.crypto.secp256k1.Secp256k1PublicKey* method), 11
 to_bytes() (*libp2p.peer.id.ID* method), 27
 to_string() (*libp2p.peer.id.ID* method), 27
 topic_ids (*libp2p.pubsub.abc.IPubsub* property), 42
 topic_ids (*libp2p.pubsub.pubsub.Pubsub* property), 50
 topic_validators (*libp2p.pubsub.pubsub.Pubsub* attribute), 50
 TopicDescriptor (class in *libp2p.pubsub.pb.rpc_pb2*), 41
 TopicDescriptor.AuthOpts (class in *libp2p.pubsub.pb.rpc_pb2*), 41
 TopicDescriptor.EncOpts (class in *libp2p.pubsub.pb.rpc_pb2*), 42
 topics (*libp2p.pubsub.mcache.CacheEntry* attribute), 47
 TopicValidator (class in *libp2p.pubsub.pubsub*), 51
 Transport (class in *libp2p.security.noise.transport*), 57
 Transport (class in *libp2p.security.secio.transport*), 60
 transport (*libp2p.network.swarm.Swarm* attribute), 26
 transport (*libp2p.tools.factories.SwarmFactory* attribute), 75

transports (*libp2p.security.security_multistream.SecurityMultistream* attribute), 64
 transports (*libp2p.stream_muxer.muxer_multistream.MuxerMultistream* attribute), 70
 TransportUpgrader (class in *libp2p.transport.upgrader*), 79
 TrioSubscriptionAPI (class in *libp2p.pubsub.subscription*), 52
 TrioTCPStream (class in *libp2p.io.trio*), 18
 try_select() (*libp2p.protocol_muxer.multiselect_client.MultiselectClient* method), 38
 try_select() (*libp2p.protocol_muxer.multiselect_client_interface.IMultiselectClient* method), 39

U

unsubscribe() (*libp2p.pubsub.abc.IPubsub* method), 42
 unsubscribe() (*libp2p.pubsub.abc.ISubscriptionAPI* method), 43
 unsubscribe() (*libp2p.pubsub.pubsub.Pubsub* method), 50
 unsubscribe() (*libp2p.pubsub.subscription.TrioSubscriptionAPI* method), 52
 unsubscribe_fn (*libp2p.pubsub.subscription.TrioSubscriptionAPI* attribute), 52
 update() (*libp2p.peer.id.IdentityHash* method), 27
 upgrade_connection() (*libp2p.transport.upgrader.TransportUpgrader* method), 79
 upgrade_listener() (*libp2p.transport.upgrader.TransportUpgrader* method), 79
 upgrade_security() (*libp2p.transport.upgrader.TransportUpgrader* method), 79
 UpgradeFailure, 77
 upgrader (*libp2p.network.swarm.Swarm* attribute), 26
 upgrader (*libp2p.tools.factories.SwarmFactory* attribute), 75

V

validate_msg() (*libp2p.pubsub.pubsub.Pubsub* method), 50
 ValidationError, 79
 validator (*libp2p.pubsub.pubsub.TopicValidator* attribute), 51
 VarIntLengthMsgReader (class in *libp2p.io.msgio*), 18
 verify() (*libp2p.crypto.ecc.ECCPublicKey* method), 7
 verify() (*libp2p.crypto.ed25519.Ed25519PublicKey* method), 8
 verify() (*libp2p.crypto.keys.PublicKey* method), 9
 verify() (*libp2p.crypto.rsa.RSAPublicKey* method), 10
 verify() (*libp2p.crypto.secp256k1.Secp256k1PublicKey* method), 11

`verify_handshake_payload_sig()` (in module `libp2p.security.noise.messages`), 56

W

`wait_until_ready()` (`libp2p.pubsub.abc.IPubsub` method), 42

`wait_until_ready()` (`libp2p.pubsub.pubsub.Pubsub` method), 50

`window()` (`libp2p.pubsub.mcache.MessageCache` method), 48

`window_size` (`libp2p.pubsub.mcache.MessageCache` attribute), 48

`with_noise_pipes` (`libp2p.security.noise.transport.Transport` attribute), 58

`write()` (`libp2p.io.abc.Writer` method), 17

`write()` (`libp2p.io.trio.TrioTCPStream` method), 18

`write()` (`libp2p.network.connection.raw_connection.RawConnection` method), 19

`write()` (`libp2p.network.stream.net_stream.NetStream` method), 21

`write()` (`libp2p.protocol_muxer.multiselect_communicator.MultiselectCommunicator` method), 40

`write()` (`libp2p.protocol_muxer.multiselect_communicator_interface.IMultiselectCommunicator` method), 40

`write()` (`libp2p.security.insecure.transport.InsecureSession` method), 54

`write()` (`libp2p.security.secure_session.SecureSession` method), 63

`write()` (`libp2p.stream_muxer.mplex.mplex_stream.MplexStream` method), 68

`write_data()` (in module `examples.chat.chat`), 5

`write_deadline` (`libp2p.stream_muxer.mplex.mplex_stream.MplexStream` attribute), 68

`write_lock` (`libp2p.io.trio.TrioTCPStream` attribute), 18

`write_msg()` (`libp2p.io.abc.MsgWriter` method), 16

`write_msg()` (`libp2p.io.msgio.BaseMsgReadWriter` method), 17

`write_msg()` (`libp2p.security.noise.io.BaseNoiseMsgReadWriter` method), 56

`write_msg()` (`libp2p.security.secio.transport.SecioMsgReadWriter` method), 60

`write_to_stream()` (`libp2p.stream_muxer.mplex.mplex.Mplex` method), 66

`WriteCloser` (class in `libp2p.io.abc`), 16

`Writer` (class in `libp2p.io.abc`), 16

X

`xor_id` (`libp2p.peer.id.ID` property), 27