
py-libp2p Documentation

Release 0.1.5

The Ethereum Foundation

Mar 26, 2020

CONTENTS

1	Contents	3
1.1	libp2p package	3
1.1.1	Subpackages	3
1.1.2	Submodules	70
1.1.3	libp2p.exceptions module	70
1.1.4	libp2p.typing module	70
1.1.5	libp2p.utils module	70
1.1.6	Module contents	70
1.2	Release Notes	71
1.2.1	libp2p v0.1.5 (2020-03-25)	71
1.2.2	libp2p v0.1.4 (2019-12-12)	72
1.2.3	libp2p v0.1.3 (2019-11-27)	72
1.2.4	v0.1.2	73
1.3	examples package	73
1.3.1	Subpackages	73
1.3.2	Module contents	73
2	Indices and tables	75
	Python Module Index	77
	Index	79

The Python implementation of the libp2p networking stack

CONTENTS

1.1 libp2p package

1.1.1 Subpackages

libp2p.crypto package

Subpackages

libp2p.crypto.pb package

Submodules

libp2p.crypto.pb.crypto_pb2 module

```
class libp2p.crypto.pb.crypto_pb2.PrivateKey
    Bases: google.protobuf.pyext._message.CMessage, google.protobuf.message.
    Message
    DESCRIPTOR = <google.protobuf.pyext._message.MessageDescriptor object>
    data
        Field crypto.pb.PrivateKey.data
    key_type
        Field crypto.pb.PrivateKey.key_type
class libp2p.crypto.pb.crypto_pb2.PublicKey
    Bases: google.protobuf.pyext._message.CMessage, google.protobuf.message.
    Message
    DESCRIPTOR = <google.protobuf.pyext._message.MessageDescriptor object>
    data
        Field crypto.pb.PublicKey.data
    key_type
        Field crypto.pb.PublicKey.key_type
```

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,   ci-
                                                                    pher_key:
                                                                    bytes)
```

Bases: `object`

cipher_key: `bytes = None`

cipher_type: `str = None`

hash_type: `str = None`

iv: `bytes = None`

mac_key: `bytes = None`

```
exception libp2p.crypto.authenticated_encryption.InvalidMACException
```

Bases: `Exception`

```
class libp2p.crypto.authenticated_encryption.MacAndCipher (parameters:
                                                                    libp2p.crypto.authenticated_encryption.Encryptio)
```

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
                                                                    [libp2p.crypto.authenticated_encryption.Encryptio
                                                                    libp2p.crypto.authenticated_encryption.EncryptionPa
```

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='140433643288952'>)
```

Bases: `libp2p.crypto.keys.PrivateKey`

get_public_key () → `libp2p.crypto.keys.PublicKey`

get_type () → `libp2p.crypto.keys.KeyType`

Returns the `KeyType` for `self`.

classmethod new (*curve: str*) → `libp2p.crypto.ecc.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='140433643309712'>,  curve:      <MagicMock
                                             name='mock.curve.Curve' id='140433643288952'>)
```

Bases: *libp2p.crypto.keys.PublicKey*

classmethod from_bytes (*data: bytes, curve: str*) → libp2p.crypto.ecc.ECCPublicKey

get_type () → libp2p.crypto.keys.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*) → libp2p.crypto.keys.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='140433643288952'>
```

converts 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: nacl.public.PrivateKey)
```

Bases: *libp2p.crypto.keys.PrivateKey*

classmethod from_bytes (*data: bytes*) → libp2p.crypto.ed25519.Ed25519PrivateKey

get_public_key () → libp2p.crypto.keys.PublicKey

get_type () → libp2p.crypto.keys.KeyType

Returns the KeyType for self.

classmethod new (*seed: bytes = None*) → libp2p.crypto.ed25519.Ed25519PrivateKey

sign (*data: bytes*) → bytes

to_bytes () → bytes

Returns the byte representation of this key.

```
class libp2p.crypto.ed25519.Ed25519PublicKey (impl: nacl.public.PublicKey)
```

Bases: *libp2p.crypto.keys.PublicKey*

classmethod from_bytes (*key_bytes: bytes*) → libp2p.crypto.ed25519.Ed25519PublicKey

get_type () → libp2p.crypto.keys.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) →
                                             libp2p.crypto.keys.KeyPair
```

libp2p.crypto.exceptions module

exception libp2p.crypto.exceptions.**CryptographyError**

Bases: *libp2p.exceptions.BaseLibp2pError*

exception libp2p.crypto.exceptions.**MissingDeserializerError**

Bases: *libp2p.crypto.exceptions.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[libp2p.crypto.keys.PublicKey, Callable[[bytes], bytes]]

Facilitates ECDH key exchange.

libp2p.crypto.keys module

class libp2p.crypto.keys.**Key**

Bases: *abc.ABC*

A Key represents a cryptographic key.

abstract **get_type** () → libp2p.crypto.keys.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 = None

public_key: PublicKey = None

class libp2p.crypto.keys.**KeyType**

Bases: *enum.Enum*

An enumeration.

ECC_P256 = 4

ECDSA = 3

Ed25519 = 1

RSA = 0

Secp256k1 = 2

class libp2p.crypto.keys.**PrivateKey**

Bases: *libp2p.crypto.keys.Key*

A PrivateKey represents a cryptographic private key.

classmethod **deserialize_from_protobuf** (*protobuf_data: bytes*) → libp2p.crypto.pb.crypto_pb2.PrivateKey

abstract **get_public_key** () → libp2p.crypto.keys.PublicKey

serialize () → bytes
Return the canonical serialization of this Key.

abstract sign (data: bytes) → bytes

class libp2p.crypto.keys.**PublicKey**

Bases: *libp2p.crypto.keys.Key*

A `PublicKey` represents a cryptographic public key.

classmethod deserialize_from_protobuf (protobuf_data: bytes) → libp2p.crypto.pb.crypto_pb2.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: *Crypto.PublicKey.RSA.RsaKey*)

Bases: *libp2p.crypto.keys.PrivateKey*

get_public_key () → libp2p.crypto.keys.PublicKey

get_type () → libp2p.crypto.keys.KeyType
Returns the `KeyType` for self.

classmethod new (bits: int = 2048, e: int = 65537) → libp2p.crypto.rsa.RSAPrivateKey

sign (data: bytes) → bytes

to_bytes () → bytes
Returns the byte representation of this key.

class libp2p.crypto.rsa.**RSAPublicKey** (impl: *Crypto.PublicKey.RSA.RsaKey*)

Bases: *libp2p.crypto.keys.PublicKey*

classmethod from_bytes (key_bytes: bytes) → libp2p.crypto.rsa.RSAPublicKey

get_type () → libp2p.crypto.keys.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) → libp2p.crypto.keys.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: coinkurve.keys.PrivateKey*)

Bases: *libp2p.crypto.keys.PrivateKey*

classmethod **deserialize** (*data: bytes*) → libp2p.crypto.secp256k1.Secp256k1PrivateKey

classmethod **from_bytes** (*data: bytes*) → libp2p.crypto.secp256k1.Secp256k1PrivateKey

get_public_key () → libp2p.crypto.keys.PublicKey

get_type () → libp2p.crypto.keys.KeyType

Returns the KeyType for self.

classmethod **new** (*secret: bytes = None*) → libp2p.crypto.secp256k1.Secp256k1PrivateKey

sign (*data: bytes*) → bytes

to_bytes () → bytes

Returns the byte representation of this key.

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

Bases: *libp2p.crypto.keys.PublicKey*

classmethod **deserialize** (*data: bytes*) → libp2p.crypto.secp256k1.Secp256k1PublicKey

classmethod **from_bytes** (*data: bytes*) → libp2p.crypto.secp256k1.Secp256k1PublicKey

get_type () → libp2p.crypto.keys.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*) → libp2p.crypto.keys.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*) → libp2p.crypto.keys.PrivateKey

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

Module contents

libp2p.host package

Submodules

libp2p.host.basic_host module

```
class libp2p.host.basic_host.BasicHost (network: libp2p.network.network_interface.INetworkService,  

                                         default_protocols: OrderedDict[TProtocol,  

                                         StreamHandlerFn] = None)
```

Bases: *libp2p.host.host_interface.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: libp2p.peer.peerinfo.PeerInfo*) → None

connect ensures 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: libp2p.peer.id.ID*) → None

get_addrs () → List[multiaddr.multiaddr.Multiaddr]

Returns all the multiaddr addresses this host is listening to

get_id () → libp2p.peer.id.ID

Returns *peer_id* of host

get_mux () → libp2p.protocol_muxer.multiselect.Multiselect

Returns mux instance of host

get_network () → libp2p.network.network_interface.INetworkService

Returns network instance of host

get_peerstore () → libp2p.peer.peerstore_interface.IPeerStore

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

get_private_key () → libp2p.crypto.keys.PrivateKey

Returns the private key belonging to the peer

get_public_key () → libp2p.crypto.keys.PublicKey

Returns the public key belonging to the peer

multiselect: Multiselect = None

multiselect_client: MultiselectClient = None

async new_stream (*peer_id: libp2p.peer.id.ID,* *protocol_ids: Sequence[NewType.<locals>.new_type]*) → libp2p.network.stream.net_stream_interface.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 = None`

run (*listen_addrs: Sequence[multiaddr.multiaddr.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: NewType.<locals>.new_type, stream_handler: Callable[[INetStream], 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: libp2p.host.host_interface.IHost*) → OrderedDict[TProtocol, StreamHandlerFn]

libp2p.host.exceptions module

exception `libp2p.host.exceptions.ConnectionFailure`
 Bases: `libp2p.host.exceptions.HostException`

exception `libp2p.host.exceptions.HostException`
 Bases: `libp2p.exceptions.BaseLibp2pError`

A generic exception in *IHost*.

exception `libp2p.host.exceptions.StreamFailure`
 Bases: `libp2p.host.exceptions.HostException`

libp2p.host.host_interface module

class `libp2p.host.host_interface.IHost`
 Bases: `abc.ABC`

abstract async close () → None

abstract async connect (*peer_info: libp2p.peer.peerinfo.PeerInfo*) → None
 connect ensures 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: libp2p.peer.id.ID*) → None

abstract get_addrs () → List[multiaddr.multiaddr.Multiaddr]

Returns all the multiaddr addresses this host is listening to

abstract `get_id()` → `libp2p.peer.id.ID`

Returns `peer_id` of host

abstract `get_mux()` → `Any`

Returns mux instance of host

abstract `get_network()` → `libp2p.network.network_interface.INetworkService`

Returns network instance of host

abstract `get_private_key()` → `libp2p.crypto.keys.PrivateKey`

Returns the private key belonging to the peer

abstract `get_public_key()` → `libp2p.crypto.keys.PublicKey`

Returns the public key belonging to the peer

abstract `async new_stream(peer_id: libp2p.peer.id.ID, protocol_ids: Sequence[NewType.<locals>.new_type])` → `libp2p.network.stream.net_stream_interface.INetStream` *See →*

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.multiaddr.Multiaddr])` → `AbstractAsyncContextManager[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: NewType.<locals>.new_type, 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: libp2p.network.stream.net_stream_interface.INetStream)` → `None`
`handle_ping` responds to incoming ping requests until one side errors or closes the stream.

libp2p.host.routed_host module

class libp2p.host.routed_host.**RoutedHost** (*network: libp2p.network.network_interface.INetworkService, router: libp2p.routing.interfaces.IPeerRouting*)

Bases: *libp2p.host.basic_host.BasicHost*

async connect (*peer_info: libp2p.peer.peerinfo.PeerInfo*) → None

connect ensures 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

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

Bases: *google.protobuf.pyext._message.CMessage, google.protobuf.message.Message*

DESCRIPTOR = *<google.protobuf.pyext._message.MessageDescriptor object>*

agent_version

Field *identify.pb.Identify.agent_version*

listen_addrs

Field *identify.pb.Identify.listen_addrs*

observed_addr

Field *identify.pb.Identify.observed_addr*

protocol_version

Field *identify.pb.Identify.protocol_version*

protocols

Field *identify.pb.Identify.protocols*

public_key

Field *identify.pb.Identify.public_key*

Module contents

Submodules

libp2p.identity.identify.protocol module

```
libp2p.identity.identify.protocol.identify_handler_for (host:
                                                    libp2p.host.host_interface.IHost)
                                                    →
                                                    Callable[[libp2p.network.stream.net_stream_interface
                                                    Awaitable[None]]]
```

Module contents

Module contents

libp2p.io package

Submodules

libp2p.io.abc module

```
class libp2p.io.abc.Closer
```

```
    Bases: abc.ABC
```

```
    abstract async close () → None
```

```
class libp2p.io.abc.EncryptedMsgReadWrite
```

```
    Bases: libp2p.io.abc.MsgReadWriteCloser, libp2p.io.abc.Encrypter
```

```
    Read/write message with encryption/decryption.
```

```
class libp2p.io.abc.Encrypter
```

```
    Bases: abc.ABC
```

```
    abstract decrypt (data: bytes) → bytes
```

```
    abstract encrypt (data: bytes) → bytes
```

```
class libp2p.io.abc.MsgReadWriteCloser
```

```
    Bases: libp2p.io.abc.MsgReader, libp2p.io.abc.MsgWriter, libp2p.io.abc.Closer
```

```
class libp2p.io.abc.MsgReader
```

```
    Bases: abc.ABC
```

```
    abstract async read_msg () → bytes
```

```
class libp2p.io.abc.MsgWriter
```

```
    Bases: abc.ABC
```

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

```
class libp2p.io.abc.ReadCloser
```

```
    Bases: libp2p.io.abc.Reader, libp2p.io.abc.Closer
```

```
class libp2p.io.abc.ReadWriteCloser
```

```
    Bases: libp2p.io.abc.Reader, libp2p.io.abc.Writer, libp2p.io.abc.Closer
```

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

class libp2p.io.abc.Reader
    Bases: abc.ABC

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

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

class libp2p.io.abc.Writer
    Bases: abc.ABC

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

libp2p.io.exceptions module

```
exception libp2p.io.exceptions.DecryptionFailedException
    Bases: libp2p.io.exceptions.MsgioException

exception libp2p.io.exceptions.IOException
    Bases: libp2p.exceptions.BaseLibp2pError

exception libp2p.io.exceptions.IncompleteReadError
    Bases: libp2p.io.exceptions.IOException

    Fewer bytes were read than requested.

exception libp2p.io.exceptions.MessageTooLarge
    Bases: libp2p.io.exceptions.MsgioException

exception libp2p.io.exceptions.MissingLengthException
    Bases: libp2p.io.exceptions.MsgioException

exception libp2p.io.exceptions.MissingMessageException
    Bases: libp2p.io.exceptions.MsgioException

exception libp2p.io.exceptions.MsgioException
    Bases: libp2p.io.exceptions.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:
                                         libp2p.io.abc.ReadWriteCloser)
    Bases: libp2p.io.abc.MsgReadWriteCloser

    async close () → None

    abstract encode_msg (msg: bytes) → bytes

    abstract async next_msg_len () → int

    async read_msg () → bytes

    read_write_closer: ReadWriteCloser = None
```

```

    size_len_bytes: int = None
    async write_msg(msg: bytes) → None
class libp2p.io.msgio.FixedSizeLenMsgReadWriter(read_write_closer:
                                                libp2p.io.abc.ReadWriteCloser)
    Bases: libp2p.io.msgio.BaseMsgReadWriter
    encode_msg(msg: bytes) → bytes
    async next_msg_len() → int
    size_len_bytes: int = None
class libp2p.io.msgio.VarIntLengthMsgReadWriter(read_write_closer:
                                                libp2p.io.abc.ReadWriteCloser)
    Bases: libp2p.io.msgio.BaseMsgReadWriter
    encode_msg(msg: bytes) → bytes
    max_msg_size: int = None
    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: libp2p.io.abc.Reader, size_len_bytes: int) → int

```

libp2p.io.utils module

```

async libp2p.io.utils.read_exactly(reader: libp2p.io.abc.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: libp2p.io.exceptions.IOException

```

libp2p.network.connection.net_connection_interface module

```
class libp2p.network.connection.net_connection_interface.INetConn
    Bases: libp2p.io.abc.Closer

    event_started: trio.Event = None

    abstract get_streams() → Tuple[libp2p.network.stream.net_stream_interface.INetStream, ...]

    muxed_conn: IMuxedConn = None

    abstract async new_stream() → libp2p.network.stream.net_stream_interface.INetStream
```

libp2p.network.connection.raw_connection module

```
class libp2p.network.connection.raw_connection.RawConnection (stream:
                                                         libp2p.io.abc.ReadWriteCloser,
                                                         initiator: bool)
    Bases: libp2p.network.connection.raw_connection_interface.IRawConnection

    async close() → None

    is_initiator: bool = None

    async read (n: int = 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 = None

    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: libp2p.io.abc.ReadWriteCloser

    A Raw Connection provides a Reader and a Writer.

    is_initiator: bool = None
```

libp2p.network.connection.swarm_connection module

```
class libp2p.network.connection.swarm_connection.SwarmConn (muxed_conn:
                                                         libp2p.stream_muxer.abc.IMuxedConn,
                                                         swarm: Swarm)
    Bases: libp2p.network.connection.net_connection_interface.INetConn

    async close() → None

    event_closed: trio.Event = None

    get_streams() → Tuple[libp2p.network.stream.net_stream.NetStream, ...]

    property is_closed

    muxed_conn: IMuxedConn = None
```

```

async new_stream() → libp2p.network.stream.net_stream.NetStream
remove_stream(stream: libp2p.network.stream.net_stream.NetStream) → None
async start() → None
streams: Set[NetStream] = None
swarm: 'Swarm' = None

```

Module contents

libp2p.network.stream package

Submodules

libp2p.network.stream.exceptions module

```

exception libp2p.network.stream.exceptions.StreamClosed
    Bases: libp2p.network.stream.exceptions.StreamError
exception libp2p.network.stream.exceptions.StreamEOF
    Bases: libp2p.network.stream.exceptions.StreamError, EOFError
exception libp2p.network.stream.exceptions.StreamError
    Bases: libp2p.io.exceptions.IOException
exception libp2p.network.stream.exceptions.StreamReset
    Bases: libp2p.network.stream.exceptions.StreamError

```

libp2p.network.stream.net_stream module

```

class libp2p.network.stream.net_stream.NetStream(muxed_stream:
    libp2p.stream_muxer.abc.IMuxedStream)
    Bases: libp2p.network.stream.net_stream_interface.INetStream
async close() → None
    close stream.
get_protocol() → NewType.<locals>.new_type
    Returns protocol id that stream runs on
muxed_stream: IMuxedStream = None
protocol_id: Optional[TProtocol] = None
async read(n: int = None) → bytes
    reads 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: NewType.<locals>.new_type) → 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: *libp2p.io.abc.ReadWriteCloser*

abstract get_protocol () → *NewType.<locals>.new_type*

Returns protocol id that stream runs on

muxed_conn: **IMuxedConn** = None

abstract async reset () → None
Close both ends of the stream.

abstract set_protocol (*protocol_id: NewType.<locals>.new_type*) → None

Parameters **protocol_id** – protocol id that stream runs on

Module contents

Submodules

libp2p.network.exceptions module

exception libp2p.network.exceptions.**SwarmException**
Bases: *libp2p.exceptions.BaseLibp2pError*

libp2p.network.network_interface module

class libp2p.network.network_interface.**INetwork**
Bases: *abc.ABC*

abstract async close () → None

abstract async close_peer (*peer_id: libp2p.peer.id.ID*) → None

connections: **Dict**[**ID**, **INetConn**] = None

abstract async dial_peer (*peer_id: libp2p.peer.id.ID*) →
libp2p.network.connection.net_connection_interface.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 () → *libp2p.peer.id.ID*

Returns the peer id

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

Parameters `multiaddrs` – one or many multiaddrs to start listening on

Returns True if at least one success

`listeners: Dict[str, IListener] = None`

abstract `async new_stream` (*peer_id: libp2p.peer.id.ID*) →
libp2p.network.stream.net_stream_interface.INetStream

Parameters

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

Returns net stream instance

`peerstore: IPeerStore = None`

abstract `register_notifee` (*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: libp2p.network.network_interface.INetwork, async_service.abc.
ServiceAPI

`connections = None`

`listeners = None`

`peerstore = None`

libp2p.network.notifee_interface module

class libp2p.network.notifee_interface.INotifee

Bases: abc.ABC

abstract `async closed_stream` (*network: INetwork, stream: libp2p.network.stream.net_stream_interface.INetStream*) → None

Parameters

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

abstract `async connected` (*network: INetwork, conn: libp2p.network.connection.net_connection_interface.INetConn*) → None

Parameters

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

abstract `async disconnected` (*network: INetwork, conn: libp2p.network.connection.net_connection_interface.INetConn*) → None

Parameters

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

abstract async listen (*network: INetwork, multiaddr: multiaddr.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.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: libp2p.network.stream.net_stream_interface.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: libp2p.peer.id.ID, peerstore: libp2p.peer.peerstore_interface.IPeerStore, upgrader: libp2p.transport.upgrader.TransportUpgrader, transport: libp2p.transport.transport_interface.ITransport)
```

Bases: `async_service.base.Service, libp2p.network.network_interface.INetworkService`

async add_conn (*muxed_conn: libp2p.stream_muxer.abc.IMuxedConn*) → `libp2p.network.connection.swarm_connection.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: libp2p.peer.id.ID*) → None

common_stream_handler: `StreamHandlerFn = None`

connections: `Dict[ID, INetConn] = None`

async dial_addr (*addr: multiaddr.multiaddr.Multiaddr, peer_id: libp2p.peer.id.ID*) → `libp2p.network.connection.net_connection_interface.INetConn`
 dial_addr 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: libp2p.peer.id.ID*) → libp2p.network.connection.net_connection_interface.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

event_listener_nursery_created: **trio.Event = None**

get_peer_id () → libp2p.peer.id.ID

Returns the peer id

async listen (**multiaddrs: multiaddr.multiaddr.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: **Optional[trio.Nursery] = None**

listeners: **Dict[str, IListener] = None**

async new_stream (*peer_id: libp2p.peer.id.ID*) → libp2p.network.stream.net_stream_interface.INetStream

Parameters **peer_id** – peer_id of destination

Raises *SwarmException* – raised when an error occurs

Returns net stream instance

notifees: **List[INotiffee] = None**

async notify_closed_stream (*stream: libp2p.network.stream.net_stream_interface.INetStream*)
→ None

async notify_connected (*conn: libp2p.network.connection.net_connection_interface.INetConn*)
→ None

async notify_disconnected (*conn: libp2p.network.connection.net_connection_interface.INetConn*)
→ None

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

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

async notify_opened_stream (*stream: libp2p.network.stream.net_stream_interface.INetStream*)
→ None

peerstore: **IPeerStore = None**

register_notiffee (*notiffee: libp2p.network.notiffee_interface.INotiffee*) → None

Parameters `notifee` – object implementing Notifee interface

Returns true if notifee registered successfully, false otherwise

remove_conn (*swarm_conn: libp2p.network.connection.swarm_connection.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 = None

set_stream_handler (*stream_handler: Callable[[INetStream], Awaitable[None]]*) → None
Set the stream handler for all incoming streams.

transport: ITransport = None

upgrader: TransportUpgrader = None

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

Module contents

libp2p.peer package

Submodules

libp2p.peer.addrbook_interface module

class `libp2p.peer.addrbook_interface.IAddrBook`

Bases: `abc.ABC`

abstract add_addr (*peer_id: libp2p.peer.id.ID, addr: multiaddr.multiaddr.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: libp2p.peer.id.ID, addrs: Sequence[multiaddr.multiaddr.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: libp2p.peer.id.ID*) → List[multiaddr.multiaddr.Multiaddr]

Parameters **peer_id** – peer to get addresses of

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

abstract **clear_addrs** (*peer_id: libp2p.peer.id.ID*) → None

Removes all previously stored addresses.

Parameters **peer_id** – peer to remove addresses of

abstract **peers_with_addrs** () → List[libp2p.peer.id.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*) → libp2p.peer.id.ID

classmethod **from_pubkey** (*key: libp2p.crypto.keys.PublicKey*) → libp2p.peer.id.ID

pretty () → str

to_base58 () → str

to_bytes () → bytes

to_string () → str

property **xor_id**

class libp2p.peer.id.IdentityHash

Bases: object

digest () → bytes

update (*input: bytes*) → None

libp2p.peer.id.**sha256_digest** (*data: Union[str, bytes]*) → bytes

libp2p.peer.peerdata module

class libp2p.peer.peerdata.PeerData

Bases: *libp2p.peer.peerdata_interface.IPeerData*

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

Parameters **addrs** – multiaddresses to add

add_privkey (*privkey: libp2p.crypto.keys.PrivateKey*) → None

Parameters **privkey** –

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

Parameters **protocols** – protocols to add

add_pubkey (*pubkey: libp2p.crypto.keys.PublicKey*) → None

Parameters **pubkey** –

addrs: List[Multiaddr] = None

clear_addrs () → None

Clear all addresses.

get_addrs () → List[multiaddr.multiaddr.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 () → libp2p.crypto.keys.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 () → libp2p.crypto.keys.PublicKey

Returns public key of the peer

Raises *PeerDataError* – if public key not found

metadata: Dict[Any, Any] = None

privkey: PrivateKey = None

protocols: List[str] = None

pubkey: PublicKey = None

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.ABC

abstract add_addrs (*addrs: Sequence[multiaddr.multiaddr.Multiaddr]*) → None**Parameters** **addrs** – multiaddresses to add**abstract add_privkey** (*privkey: libp2p.crypto.keys.PrivateKey*) → None**Parameters** **privkey** –**abstract add_protocols** (*protocols: Sequence[str]*) → None**Parameters** **protocols** – protocols to add**abstract add_pubkey** (*pubkey: libp2p.crypto.keys.PublicKey*) → None**Parameters** **pubkey** –**abstract clear_addrs** () → None

Clear all addresses.

abstract get_addrs () → List[multiaddr.multiaddr.Multiaddr]**Returns** all multiaddresses**abstract get_metadata** (*key: str*) → libp2p.peer.peermetadata_interface.IPeerMetadata**Parameters** **key** – key in KV pair**Returns** val for key**Raises** **PeerDataError** – key not found**abstract get_privkey** () → libp2p.crypto.keys.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** () → libp2p.crypto.keys.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: libp2p.peer.id.ID, addrs: Sequence[multiaddr.multiaddr.Multiaddr]*)

Bases: object

addrs: List[multiaddr.Multiaddr] = None

peer_id: ID = None

libp2p.peer.peerinfo.info_from_p2p_addr (*addr: multiaddr.multiaddr.Multiaddr*) → libp2p.peer.peerinfo.PeerInfo

libp2p.peer.peermetadata_interface module

class libp2p.peer.peermetadata_interface.IPeerMetadata

Bases: abc.ABC

abstract get (*peer_id: libp2p.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: libp2p.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: libp2p.peer.peerstore_interface.IPeerStore

add_addr (*peer_id: libp2p.peer.id.ID, addr: multiaddr.multiaddr.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: libp2p.peer.id.ID, addrs: Sequence[multiaddr.multiaddr.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: libp2p.peer.id.ID, key_pair: libp2p.crypto.keys.KeyPair*) → None

Parameters

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

add_privkey (*peer_id: libp2p.peer.id.ID, privkey: libp2p.crypto.keys.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: libp2p.peer.id.ID, protocols: Sequence[str]*) → None

Parameters

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

add_pubkey (*peer_id: libp2p.peer.id.ID, pubkey: libp2p.crypto.keys.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: libp2p.peer.id.ID*) → List[multiaddr.multiaddr.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: libp2p.peer.id.ID*) → None

Parameters **peer_id** – peer ID to clear addrs for

get (*peer_id: libp2p.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: libp2p.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] = None

peer_ids () → List[libp2p.peer.id.ID]

Returns all of the peer IDs stored in peer store

peer_info (*peer_id: libp2p.peer.id.ID*) → libp2p.peer.peerinfo.PeerInfo

Parameters **peer_id** – peer ID to get info for

Returns peer info object

peers_with_addrs () → List[libp2p.peer.id.ID]

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

privkey (*peer_id: libp2p.peer.id.ID*) → libp2p.crypto.keys.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: libp2p.peer.id.ID*) → libp2p.crypto.keys.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: libp2p.peer.id.ID, key: str, val: Any*) → None

Parameters

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

set_protocols (*peer_id: libp2p.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: *libp2p.peer.addrbook_interface.IAddrBook*, *libp2p.peer.peermetadata_interface.IPeerMetadata***abstract add_addr** (*peer_id: libp2p.peer.id.ID*, *addr: multiaddr.multiaddr.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: libp2p.peer.id.ID*, *addrs: Sequence[multiaddr.multiaddr.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: libp2p.peer.id.ID*, *key_pair: libp2p.crypto.keys.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: libp2p.peer.id.ID*, *privkey: libp2p.crypto.keys.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: libp2p.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: libp2p.peer.id.ID*, *pubkey: libp2p.crypto.keys.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: libp2p.peer.id.ID*) → List[multiaddr.multiaddr.Multiaddr]

Parameters `peer_id` – peer ID to get addrs for

Returns list of addrs

abstract `clear_addrs` (*peer_id: libp2p.peer.id.ID*) → None

Parameters `peer_id` – peer ID to clear addrs for

abstract `get` (*peer_id: libp2p.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: libp2p.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[libp2p.peer.id.ID]

Returns all of the peer IDs stored in peer store

abstract `peer_info` (*peer_id: libp2p.peer.id.ID*) → libp2p.peer.peerinfo.PeerInfo

Parameters `peer_id` – peer ID to get info for

Returns peer info object

abstract `peers_with_addrs` () → List[libp2p.peer.id.ID]

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

abstract `privkey` (*peer_id: libp2p.peer.id.ID*) → libp2p.crypto.keys.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: libp2p.peer.id.ID*) → libp2p.crypto.keys.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: libp2p.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: libp2p.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: *libp2p.exceptions.BaseLibp2pError*

Raised when an error occurs in protocol selection process.

exception libp2p.protocol_muxer.exceptions.**MultiselectCommunicatorError**

Bases: *libp2p.exceptions.BaseLibp2pError*

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

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

Bases: *libp2p.exceptions.BaseLibp2pError*

Raised when an error occurs in multiselect process.

libp2p.protocol_muxer.multiselect module

class libp2p.protocol_muxer.multiselect.**Multiselect** (*default_handlers:*
Dict[NewType.<locals>.new_type,
Callable[[INetStream], Await-
able[None]]] = None)

Bases: *libp2p.protocol_muxer.multiselect_muxer_interface.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: NewType.<locals>.new_type, handler: Callable[[INetStream], Await-*
able[None]]) → None

Store the handler with the given protocol.

Parameters

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

handlers: **Dict**[**TProtocol**, **StreamHandlerFn**] = **None**

async handshake (*communicator: libp2p.protocol_muxer.multiselect_communicator_interface.IMultiselectCommunicator*)
 → None

Perform handshake to agree on multiselect protocol.

Parameters **communicator** – communicator to use

Raises **MultiselectError** – raised when handshake failed

async negotiate (*communicator: libp2p.protocol_muxer.multiselect_communicator_interface.IMultiselectCommunicator*)
→ Tuple[NewType.<locals>.new_type, Callable[[libp2p.network.stream.net_stream_interface.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: *libp2p.protocol_muxer.multiselect_client_interface.IMultiselectClient*

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

async handshake (*communicator: libp2p.protocol_muxer.multiselect_communicator_interface.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[NewType.<locals>.new_type], communicator: libp2p.protocol_muxer.multiselect_communicator_interface.IMultiselectCommunicator*)
→ NewType.<locals>.new_type

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: libp2p.protocol_muxer.multiselect_communicator_interface.IMultiselectCommunicator, protocol: NewType.<locals>.new_type*) → NewType.<locals>.new_type

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.ABC`

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

async handshake (*communicator: libp2p.protocol_muxer.multiselect_communicator_interface.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[NewType.<locals>.new_type], communicator: libp2p.protocol_muxer.multiselect_communicator_interface.IMultiselectCommunicator*) → `NewType.<locals>.new_type`

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

async try_select (*communicator: libp2p.protocol_muxer.multiselect_communicator_interface.IMultiselectCommunicator, protocol: NewType.<locals>.new_type*) → `NewType.<locals>.new_type`

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:
                                                                    libp2p.io.abc.ReadV
Bases:
    libp2p.protocol_muxer.multiselect_communicator_interface.
    IMultiselectCommunicator

async read() → str

    Raises MultiselectCommunicatorError – raised when failed to read from underlying
    reader

read_writer: ReadWriteCloser = None

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.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.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: NewType.<locals>.new_type, handler: Callable[[INetStream],
                                                                    Awaitable[None]]) → None
    Store the handler with the given protocol.

    Parameters
        • protocol – protocol name
        • handler – handler function

get_protocols() → Tuple[NewType.<locals>.new_type, ...]

handlers: Dict[TProtocol, StreamHandlerFn] = None
```

```
abstract async negotiate (communicator: libp2p.protocol_muxer.multiselect_communicator_interface.IMultiselectCom
    → Tuple[NewType.<locals>.new_type,
    Callable[[libp2p.network.stream.net_stream_interface.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

```
class libp2p.pubsub.pb.rpc_pb2.ControlGraft
```

```
  Bases: google.protobuf.pyext._message.CMessage, google.protobuf.message.Message
```

```
  DESCRIPTOR = <google.protobuf.pyext._message.MessageDescriptor object>
```

```
  topicID
```

```
    Field pubsub.pb.ControlGraft.topicID
```

```
class libp2p.pubsub.pb.rpc_pb2.ControlIHave
```

```
  Bases: google.protobuf.pyext._message.CMessage, google.protobuf.message.Message
```

```
  DESCRIPTOR = <google.protobuf.pyext._message.MessageDescriptor object>
```

```
  messageIDs
```

```
    Field pubsub.pb.ControlIHave.messageIDs
```

```
  topicID
```

```
    Field pubsub.pb.ControlIHave.topicID
```

```
class libp2p.pubsub.pb.rpc_pb2.ControlIWant
```

```
  Bases: google.protobuf.pyext._message.CMessage, google.protobuf.message.Message
```

```
  DESCRIPTOR = <google.protobuf.pyext._message.MessageDescriptor object>
```

```
  messageIDs
```

```
    Field pubsub.pb.ControlIWant.messageIDs
```

```
class libp2p.pubsub.pb.rpc_pb2.ControlMessage
```

```
  Bases: google.protobuf.pyext._message.CMessage, google.protobuf.message.Message
```

```
  DESCRIPTOR = <google.protobuf.pyext._message.MessageDescriptor object>
```

graft
Field pubsub.pb.ControlMessage.graft

ihave
Field pubsub.pb.ControlMessage.ihave

iwant
Field pubsub.pb.ControlMessage.iwant

prune
Field pubsub.pb.ControlMessage.prune

class libp2p.pubsub.pb.rpc_pb2.**ControlPrune**
Bases: google.protobuf.pyext._message.CMessage, google.protobuf.message.Message

DESCRIPTOR = <google.protobuf.pyext._message.MessageDescriptor object>

topicID
Field pubsub.pb.ControlPrune.topicID

class libp2p.pubsub.pb.rpc_pb2.**Message**
Bases: google.protobuf.pyext._message.CMessage, google.protobuf.message.Message

DESCRIPTOR = <google.protobuf.pyext._message.MessageDescriptor object>

data
Field pubsub.pb.Message.data

from_id
Field pubsub.pb.Message.from_id

key
Field pubsub.pb.Message.key

seqno
Field pubsub.pb.Message.seqno

signature
Field pubsub.pb.Message.signature

topicIDs
Field pubsub.pb.Message.topicIDs

class libp2p.pubsub.pb.rpc_pb2.**RPC**
Bases: google.protobuf.pyext._message.CMessage, google.protobuf.message.Message

DESCRIPTOR = <google.protobuf.pyext._message.MessageDescriptor object>

class SubOpts
Bases: google.protobuf.pyext._message.CMessage, google.protobuf.message.Message

DESCRIPTOR = <google.protobuf.pyext._message.MessageDescriptor object>

subscribe
Field pubsub.pb.RPC.SubOpts.subscribe

topicid
Field pubsub.pb.RPC.SubOpts.topicid

control
Field pubsub.pb.RPC.control

publish
Field pubsub.pb.RPC.publish

subscriptions
Field pubsub.pb.RPC.subscriptions

class libp2p.pubsub.pb.rpc_pb2.TopicDescriptor

Bases: google.protobuf.pyext._message.CMessage, google.protobuf.message.Message

class AuthOpts

Bases: google.protobuf.pyext._message.CMessage, google.protobuf.message.Message

AuthMode = <google.protobuf.internal.enum_type_wrapper.EnumTypeWrapper object>

DESCRIPTOR = <google.protobuf.pyext._message.MessageDescriptor object>

KEY = 1

NONE = 0

WOT = 2

keys

Field pubsub.pb.TopicDescriptor.AuthOpts.keys

mode

Field pubsub.pb.TopicDescriptor.AuthOpts.mode

DESCRIPTOR = <google.protobuf.pyext._message.MessageDescriptor object>

class EncOpts

Bases: google.protobuf.pyext._message.CMessage, google.protobuf.message.Message

DESCRIPTOR = <google.protobuf.pyext._message.MessageDescriptor object>

EncMode = <google.protobuf.internal.enum_type_wrapper.EnumTypeWrapper object>

NONE = 0

SHAREDKEY = 1

WOT = 2

keyHashes

Field pubsub.pb.TopicDescriptor.EncOpts.keyHashes

mode

Field pubsub.pb.TopicDescriptor.EncOpts.mode

auth

Field pubsub.pb.TopicDescriptor.auth

enc

Field pubsub.pb.TopicDescriptor.enc

name

Field pubsub.pb.TopicDescriptor.name

Module contents

Submodules

libp2p.pubsub.abc module

class libp2p.pubsub.abc.IPubsub

Bases: `async_service.abc.ServiceAPI`

abstract property `my_id`

abstract property `protocols`

abstract async publish (`topic_id: str, data: bytes`) → None

abstract remove_topic_validator (`topic: str`) → None

abstract set_topic_validator (`topic: str, validator: Union[Callable[[libp2p.peer.id.ID, libp2p.pubsub.pb.rpc_pb2.Message], bool], Callable[[libp2p.peer.id.ID, libp2p.pubsub.pb.rpc_pb2.Message], Awaitable[bool]]], is_async_validator: bool`) → None

abstract async subscribe (`topic_id: str`) → libp2p.pubsub.abc.ISubscriptionAPI

abstract property `topic_ids`

abstract async unsubscribe (`topic_id: str`) → None

abstract async wait_until_ready () → None

class libp2p.pubsub.abc.IPubsubRouter

Bases: `abc.ABC`

abstract add_peer (`peer_id: libp2p.peer.id.ID, protocol_id: NewType.<locals>.new_type`) → 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[NewType.<locals>.new_type]

Returns the list of protocols supported by the router

abstract async handle_rpc (`rpc: libp2p.pubsub.pb.rpc_pb2.RPC, sender_peer_id: libp2p.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: libp2p.peer.id.ID, pubsub_msg: libp2p.pubsub.pb.rpc_pb2.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: libp2p.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: contextlib.AbstractAsyncContextManager, collections.abc.AsyncIterable, typing.Generic

abstract async get () → libp2p.pubsub.pb.rpc_pb2.Message

abstract async unsubscribe () → None

libp2p.pubsub.exceptions module

exception libp2p.pubsub.exceptions.**NoPubsubAttached**
 Bases: *libp2p.pubsub.exceptions.PubsubRouterError*

exception libp2p.pubsub.exceptions.**PubsubRouterError**
 Bases: *libp2p.exceptions.BaseLibp2pError*

libp2p.pubsub.floodsub module

class libp2p.pubsub.floodsub.**FloodSub** (*protocols: Sequence[NewType.<locals>.new_type]*)
 Bases: *libp2p.pubsub.abc.IPubsubRouter*

add_peer (*peer_id: libp2p.peer.id.ID, protocol_id: NewType.<locals>.new_type*) → None
 Notifies the router that a new peer has been connected.

Parameters **peer_id** – id of peer to add

attach (*pubsub: libp2p.pubsub.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[NewType.<locals>.new_type]

Returns the list of protocols supported by the router

async handle_rpc (*rpc: libp2p.pubsub.pb.rpc_pb2.RPC, sender_peer_id: libp2p.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] = None

async publish (*msg_forwarder: libp2p.peer.id.ID, pubsub_msg: libp2p.pubsub.pb.rpc_pb2.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 = None

remove_peer (*peer_id: libp2p.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[NewType.<locals>.new_type], 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: *libp2p.pubsub.abc.IPubsubRouter*, *async_service.base.Service*

add_peer (*peer_id: libp2p.peer.id.ID, protocol_id: NewType.<locals>.new_type*) → 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: libp2p.pubsub.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 = None

degree_high: int = None

degree_low: int = None

async emit_control_message (*control_msg: libp2p.pubsub.pb.rpc_pb2.ControlMessage, to_peer: libp2p.peer.id.ID*) → None

async emit_graft (*topic: str, to_peer: libp2p.peer.id.ID*) → None

Emit graft message, sent to *to_peer*, for *topic*.

async emit_ihave (*topic: str, msg_ids: Any, to_peer: libp2p.peer.id.ID*) → None

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

async emit_iwant (*msg_ids: Any, to_peer: libp2p.peer.id.ID*) → None

Emit iwant message, sent to *to_peer*, for *msg_ids*.

async emit_prune (*topic: str, to_peer: libp2p.peer.id.ID*) → None

Emit graft message, sent to *to_peer*, for *topic*.

fanout: Dict[str, Set[ID]] = None

fanout_heartbeat () → None

get_protocols () → List[NewType.<locals>.new_type]

Returns the list of protocols supported by the router

gossip_heartbeat () → DefaultDict[libp2p.peer.id.ID, Dict[str, List[str]]]

async handle_graft (*graft_msg: libp2p.pubsub.pb.rpc_pb2.ControlGraft, sender_peer_id: libp2p.peer.id.ID*) → None

async handle_ihave (*ihave_msg: libp2p.pubsub.pb.rpc_pb2.ControlIHave, sender_peer_id: libp2p.peer.id.ID*) → None

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

async handle_iwant (*iwant_msg: libp2p.pubsub.pb.rpc_pb2.ControlIWant, sender_peer_id: libp2p.peer.id.ID*) → None

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

async handle_prune (*prune_msg: libp2p.pubsub.pb.rpc_pb2.ControlPrune, sender_peer_id: libp2p.peer.id.ID*) → None

async handle_rpc (*rpc: libp2p.pubsub.pb.rpc_pb2.RPC, sender_peer_id: libp2p.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 = None

heartbeat_interval: int = None

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 = None`

mesh: `Dict[str, Set[ID]] = None`

mesh_heartbeat () → Tuple[DefaultDict[libp2p.peer.id.ID, List[str]], DefaultDict[libp2p.peer.id.ID, List[str]]]

pack_control_msgs (*ihave_msgs:* `List[libp2p.pubsub.pb.rpc_pb2.ControlIHave]`,
graft_msgs: `List[libp2p.pubsub.pb.rpc_pb2.ControlGraft]`,
prune_msgs: `List[libp2p.pubsub.pb.rpc_pb2.ControlPrune]`) →
`libp2p.pubsub.pb.rpc_pb2.ControlMessage`

peer_protocol: `Dict[ID, TProtocol] = None`

protocols: `List[TProtocol] = None`

async publish (*msg_forwarder:* `libp2p.peer.id.ID`, *pubsub_msg:*
`libp2p.pubsub.pb.rpc_pb2.Message`) → None
 Invoked to forward a new message that has been validated.

pubsub: `Pubsub = None`

remove_peer (*peer_id:* `libp2p.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 (ex-
 cluding elements in *minus*) :param *minus:* elements to be excluded from selection pool :return: list of
 selected elements

time_to_live: `int = None`

libp2p.pubsub.mcache module

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

mid: `Tuple[bytes, bytes] = None`

topics: `List[str] = None`

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]`) → `Optional[libp2p.pubsub.pb.rpc_pb2.Message]`
 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]] = None

history_size: int = None

msgs: Dict[Tuple[bytes, bytes], rpc_pb2.Message] = None

put (*msg: libp2p.pubsub.pb.rpc_pb2.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 = None

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: Callable[[libp2p.pubsub.pb.rpc_pb2.Message], bytes] = <function get_peer_and_seqno_msg_id>)
```

Bases: `async_service.base.Service`, `libp2p.pubsub.abc.IPubsub`

async continuously_read_stream (*stream: libp2p.network.stream.net_stream_interface.INetStream*) → None

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

Parameters stream – stream to continuously read from

counter: int = None

dead_peer_receive_channel: 'trio.MemoryReceiveChannel[ID]' = None

event_handle_dead_peer_queue_started: trio.Event = None

event_handle_peer_queue_started: trio.Event = None

get_hello_packet () → libp2p.pubsub.pb.rpc_pb2.RPC

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

get_msg_validators (*msg: libp2p.pubsub.pb.rpc_pb2.Message*) → Tuple[libp2p.pubsub.pubsub.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*: *libp2p.peer.id.ID*, *sub_message*: *libp2p.pubsub.pb.rpc_pb2.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 = None

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

notify_subscriptions (*publish_message*: *libp2p.pubsub.pb.rpc_pb2.Message*) → None

Put incoming message from a peer onto my blocking queue.

Parameters publish_message – RPC.Message format

peer_receive_channel: 'trio.MemoryReceiveChannel[ID]' = None

peer_topics: Dict[str, Set[ID]] = None

peers: Dict[ID, INetStream] = None

property protocols

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*: *libp2p.peer.id.ID*, *msg*: *libp2p.pubsub.pb.rpc_pb2.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' = None

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 = None

set_topic_validator (*topic:* str, *validator:* Union[Callable[[libp2p.peer.id.ID, libp2p.pubsub.pb.rpc_pb2.Message], bool], Callable[[libp2p.peer.id.ID, libp2p.pubsub.pb.rpc_pb2.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 = None

async stream_handler (*stream:* libp2p.network.stream.net_stream_interface.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 = None

async subscribe (*topic_id:* str) → libp2p.pubsub.abcs.ISubscriptionAPI

Subscribe ourselves to a topic.

Parameters **topic_id** – topic_id to subscribe to

subscribed_topics_receive: Dict[str, 'TrioSubscriptionAPI'] = None

subscribed_topics_send: Dict[str, 'trio.MemorySendChannel[rpc_pb2.Message]'] = None

property **topic_ids**

topic_validators: Dict[str, TopicValidator] = None

async unsubscribe (*topic_id:* str) → None

Unsubscribe ourselves from a topic.

Parameters **topic_id** – topic_id to unsubscribe from

async validate_msg (*msg_forwarder:* libp2p.peer.id.ID, *msg:* libp2p.pubsub.pb.rpc_pb2.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: tuple

property **is_async**

Alias for field number 1

property validator

Alias for field number 0

`libp2p.pubsub.pubsub.get_content_addressed_msg_id` (*msg*:
libp2p.pubsub.pb.rpc_pb2.Message)
 → bytes

`libp2p.pubsub.pubsub.get_peer_and_seqno_msg_id` (*msg*: *libp2p.pubsub.pb.rpc_pb2.Message*)
 → bytes

libp2p.pubsub.pubsub_notifee module

class `libp2p.pubsub.pubsub_notifee.PubsubNotifee` (*initiator_peers_queue*:
trio.MemorySendChannel[ID],
dead_peers_queue:
trio.MemorySendChannel[ID])

Bases: *libp2p.network.notifee_interface.INotifee*

async `closed_stream` (*network*: *libp2p.network.network_interface.INetwork*, *stream*:
libp2p.network.stream.net_stream_interface.INetStream) → None

Parameters

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

async `connected` (*network*: *libp2p.network.network_interface.INetwork*, *conn*:
libp2p.network.connection.net_connection_interface.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]' = None`

async `disconnected` (*network*: *libp2p.network.network_interface.INetwork*, *conn*:
libp2p.network.connection.net_connection_interface.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]' = None`

async `listen` (*network*: *libp2p.network.network_interface.INetwork*, *multiaddr*: *multi-addr.multiaddr.Multiaddr*) → None

Parameters

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

async `listen_close` (*network*: *libp2p.network.network_interface.INetwork*, *multiaddr*: *multi-addr.multiaddr.Multiaddr*) → None

Parameters

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

async opened_stream (*network*: *libp2p.network.network_interface.INetwork*, *stream*: *libp2p.network.stream.net_stream_interface.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: *libp2p.pubsub.abc.ISubscriptionAPI*

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

Bases: *libp2p.pubsub.subscription.BaseSubscriptionAPI*

async get () → libp2p.pubsub.pb.rpc_pb2.Message

receive_channel: 'trio.MemoryReceiveChannel[rpc_pb2.Message]' = None

async unsubscribe () → None

unsubscribe_fn: UnsubscribeFn = None

libp2p.pubsub.validators module

libp2p.pubsub.validators.**signature_validator** (*msg*: *libp2p.pubsub.pb.rpc_pb2.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.ABC*

abstract find_provider_iter (*cid: bytes, count: int*) → Iterable[libp2p.peer.peerinfo.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.ABC

abstract async find_peer (*peer_id: libp2p.peer.id.ID*) → libp2p.peer.peerinfo.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

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

Bases: google.protobuf.pyext._message.CMessage, google.protobuf.message.Message

DESCRIPTOR = <google.protobuf.pyext._message.MessageDescriptor object>

id

Field plaintext.pb.Exchange.id

pubkey

Field plaintext.pb.Exchange.pubkey

Module contents

Submodules

libp2p.security.insecure.transport module

```
class libp2p.security.insecure.transport.InsecureSession(*, local_peer:
libp2p.peer.id.ID,
local_private_key:
libp2p.crypto.keys.PrivateKey,
remote_peer:
libp2p.peer.id.ID, re-
mote_permanent_pubkey:
libp2p.crypto.keys.PublicKey,
is_initiator: bool, conn:
libp2p.io.abc.ReadWriteCloser)
```

Bases: *libp2p.security.base_session.BaseSession*

async **close**() → None

local_peer = None

local_private_key = None

async **read**(n: int = None) → bytes

remote_peer = None

remote_permanent_pubkey = None

async **write**(data: bytes) → None

```
class libp2p.security.insecure.transport.InsecureTransport(local_key_pair:
libp2p.crypto.keys.KeyPair,
se-
cure_bytes_provider:
Callable[[int], bytes]
= <function de-
fault_secure_bytes_provider>)
```

Bases: *libp2p.security.base_transport.BaseSecureTransport*

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

async **secure_inbound**(conn: *libp2p.network.connection.raw_connection_interface.IRawConnection*) → *libp2p.security.secure_conn_interface.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: *libp2p.network.connection.raw_connection_interface.IRawConnection*, peer_id: *libp2p.peer.id.ID*) → *libp2p.security.secure_conn_interface.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:
libp2p.io.abc.ReadWriteCloser)
```

Bases: *libp2p.io.msgio.VarIntLengthMsgReadWriter*

```
max_msg_size: int = 65536
```

```
libp2p.security.insecure.transport.make_exchange_message (pubkey:  
    libp2p.crypto.keys.PublicKey)  
    →  
    libp2p.security.insecure.pb.plaintext_pb2.ExchangeMessage  
  
async libp2p.security.insecure.transport.run_handshake (local_peer:  
    libp2p.peer.id.ID,  
    local_private_key:  
    libp2p.crypto.keys.PrivateKey,  
    conn:  
    libp2p.network.connection.raw_connection_interface.RawConnectionInterface,  
    is_initiator: bool,  
    remote_peer_id:  
    libp2p.peer.id.ID) →  
    libp2p.security.secure_conn_interface.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

```
class libp2p.security.noise.pb.noise_pb2.NoiseHandshakePayload  
    Bases: google.protobuf.pyext._message.CMessage, google.protobuf.message.Message  
  
    DESCRIPTOR = <google.protobuf.pyext._message.MessageDescriptor object>  
  
    data  
        Field pb.NoiseHandshakePayload.data  
  
    identity_key  
        Field pb.NoiseHandshakePayload.identity_key  
  
    identity_sig  
        Field pb.NoiseHandshakePayload.identity_sig
```

Module contents

Submodules

libp2p.security.noise.exceptions module

exception libp2p.security.noise.exceptions.**HandshakeHasNotFinished**

Bases: *libp2p.security.noise.exceptions.NoiseFailure*

exception libp2p.security.noise.exceptions.**InvalidSignature**

Bases: *libp2p.security.noise.exceptions.NoiseFailure*

exception libp2p.security.noise.exceptions.**NoiseFailure**

Bases: *libp2p.security.exceptions.HandshakeFailure*

exception libp2p.security.noise.exceptions.**NoiseStateError**

Bases: *libp2p.security.noise.exceptions.NoiseFailure*

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

exception libp2p.security.noise.exceptions.**PeerIDMismatchesPubkey**

Bases: *libp2p.security.noise.exceptions.NoiseFailure*

libp2p.security.noise.io module

class libp2p.security.noise.io.**BaseNoiseMsgReadWriter** (*conn:*
libp2p.network.connection.raw_connection_interface.
noise_state:
noise.connection.NoiseConnection)

Bases: *libp2p.io.abc.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: **NoiseState** = None

prefix: **bytes** = b'\x00'

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

read_writer: **MsgReadWriteCloser** = None

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

class libp2p.security.noise.io.**NoiseHandshakeReadWriter** (*conn:*
libp2p.network.connection.raw_connection_interface.
noise_state:
noise.connection.NoiseConnection)

Bases: *libp2p.security.noise.io.BaseNoiseMsgReadWriter*

decrypt (*data: bytes*) → bytes

encrypt (*data: bytes*) → bytes

noise_state = None

read_writer = None

```

class libp2p.security.noise.io.NoisePacketReadWriter (read_write_closer:
                                                    libp2p.io.abc.ReadWriteCloser)
    Bases: libp2p.io.msgio.FixedSizeLenMsgReadWriter
    size_len_bytes: int = 2

class libp2p.security.noise.io.NoiseTransportReadWriter (conn:
                                                         libp2p.network.connection.raw_connection_interfa
                                                         noise_state:
                                                         noise.connection.NoiseConnection)
    Bases: libp2p.security.noise.io.BaseNoiseMsgReadWriter
    decrypt (data: bytes) → bytes
    encrypt (data: bytes) → bytes
    noise_state = None
    read_writer = None

```

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) → libp2p.security.noise.messages.NoiseHandshakePayload
    early_data: bytes = None
    id_pubkey: PublicKey = None
    id_sig: bytes = None
    serialize () → bytes

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

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

libp2p.security.noise.messages.verify_handshake_payload_sig (payload:
                                                             libp2p.security.noise.messages.NoiseHandsho
                                                             noise_static_pubkey:
                                                             libp2p.crypto.keys.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: libp2p.security.noise.patterns.IPattern

    create_noise_state () → noise.connection.NoiseConnection

    early_data: bytes = None

    libp2p_privkey: PrivateKey = None

    local_peer: ID = None

    make_handshake_payload () → libp2p.security.noise.messages.NoiseHandshakePayload

    noise_static_key: PrivateKey = None

    protocol_name: bytes = None

class libp2p.security.noise.patterns.IPattern
    Bases: abc.ABC

    abstract async handshake_inbound (conn: libp2p.network.connection.raw_connection_interface.IRawConnection)
        → libp2p.security.secure_conn_interface.ISecureConn

    abstract async handshake_outbound (conn: libp2p.network.connection.raw_connection_interface.IRawConnection,
        remote_peer: libp2p.peer.id.ID) →
        libp2p.security.secure_conn_interface.ISecureConn

class libp2p.security.noise.patterns.PatternXX (local_peer: libp2p.peer.id.ID,
        libp2p_privkey:
            libp2p.crypto.keys.PrivateKey,
        noise_static_key:
            libp2p.crypto.keys.PrivateKey,
        early_data: bytes = None)

    Bases: libp2p.security.noise.patterns.BasePattern

    early_data = None

    async handshake_inbound (conn: libp2p.network.connection.raw_connection_interface.IRawConnection)
        → libp2p.security.secure_conn_interface.ISecureConn

    async handshake_outbound (conn: libp2p.network.connection.raw_connection_interface.IRawConnection,
        remote_peer: libp2p.peer.id.ID) →
        libp2p.security.secure_conn_interface.ISecureConn

    libp2p_privkey = None

    local_peer = None

    noise_static_key = None

    protocol_name = None

```

libp2p.security.noise.transport module

```
class libp2p.security.noise.transport.Transport (libp2p_keypair:  
                                             libp2p.crypto.keys.KeyPair,  
                                             noise_privkey:  
                                             libp2p.crypto.keys.PrivateKey =  
                                             None, early_data: bytes = None,  
                                             with_noise_pipes: bool = False)  
  
Bases: libp2p.security.secure_transport_interface.ISecureTransport  
  
early_data: bytes = None  
get_pattern () → libp2p.security.noise.patterns.IPattern  
libp2p_privkey: PrivateKey = None  
local_peer: ID = None  
noise_privkey: PrivateKey = None  
async secure_inbound (conn: libp2p.network.connection.raw_connection_interface.IRawConnection)  
                      → libp2p.security.secure_conn_interface.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: libp2p.network.connection.raw_connection_interface.IRawConnection,  
                        peer_id: libp2p.peer.id.ID) → libp2p.security.secure_conn_interface.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 = None
```

Module contents

libp2p.security.secio package

Subpackages

libp2p.security.secio.pb package

Submodules

libp2p.security.secio.pb.spice_pb2 module

```
class libp2p.security.secio.pb.spice_pb2.Exchange  
Bases: google.protobuf.pyext._message.CMessage, google.protobuf.message.  
Message  
  
DESCRIPTOR = <google.protobuf.pyext._message.MessageDescriptor object>  
ephemeral_public_key  
Field spice_pb2.Exchange.ephemeral_public_key
```

signatureField `spipe.pb.Exchange.signature`**class** `libp2p.security.secio.pb.spipe_pb2.Propose`Bases: `google.protobuf.pyext._message.CMessage`, `google.protobuf.message.Message`**DESCRIPTOR** = `<google.protobuf.pyext._message.MessageDescriptor object>`**ciphers**Field `spipe.pb.Propose.ciphers`**exchanges**Field `spipe.pb.Propose.exchanges`**hashes**Field `spipe.pb.Propose.hashes`**public_key**Field `spipe.pb.Propose.public_key`**rand**Field `spipe.pb.Propose.rand`**Module contents****Submodules****libp2p.security.secio.exceptions module****exception** `libp2p.security.secio.exceptions.IncompatibleChoices`Bases: `libp2p.security.secio.exceptions.SecioException`**exception** `libp2p.security.secio.exceptions.InconsistentNonce`Bases: `libp2p.security.secio.exceptions.SecioException`**exception** `libp2p.security.secio.exceptions.InvalidSignatureOnExchange`Bases: `libp2p.security.secio.exceptions.SecioException`**exception** `libp2p.security.secio.exceptions.PeerMismatchException`Bases: `libp2p.security.secio.exceptions.SecioException`**exception** `libp2p.security.secio.exceptions.SecioException`Bases: `libp2p.security.exceptions.HandshakeFailure`**exception** `libp2p.security.secio.exceptions.SedesException`Bases: `libp2p.security.secio.exceptions.SecioException`**exception** `libp2p.security.secio.exceptions.SelfEncryption`Bases: `libp2p.security.secio.exceptions.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 = None`**curve_type:** `str = None`**ephemeral_public_key:** `PublicKey = None`**hash_type:** `str = None`**permanent_public_key:** `PublicKey = None`**class** libp2p.security.secio.transport.**Proposal** (*nonce:* `bytes`, *public_key:* `libp2p.crypto.keys.PublicKey`, *exchanges:* `str = 'P-256'`, *ciphers:* `str = 'AES-128'`, *hashes:* `str = 'SHA256'`)Bases: `object`

A `Proposal` represents the set of session parameters one peer in a pair of peers attempting to negotiate a *secio* channel prefers.

calculate_peer_id() → `libp2p.peer.id.ID`**ciphers:** `str = 'AES-128'`**classmethod deserialize** (*protobuf_bytes:* `bytes`) → `libp2p.security.secio.transport.Proposal`**exchanges:** `str = 'P-256'`**hashes:** `str = 'SHA256'`**nonce:** `bytes = None`**public_key:** `PublicKey = None`**serialize**() → `bytes`**class** libp2p.security.secio.transport.**SecioMsgReadWriter** (*local_encryption_parameters:* `libp2p.crypto.authenticated_encryption.EncryptionParameters`, *remote_encryption_parameters:* `libp2p.crypto.authenticated_encryption.EncryptionParameters`, *read_writer:* `libp2p.security.secio.transport.SecioPacketReadWriter`)Bases: `libp2p.io.abc.EncryptedMsgReadWriter`**async close**() → `None`**decrypt** (*data:* `bytes`) → `bytes`**encrypt** (*data:* `bytes`) → `bytes`**async read_msg**() → `bytes`**read_writer:** `SecioPacketReadWriter = None`**async write_msg** (*msg:* `bytes`) → `None`**class** libp2p.security.secio.transport.**SecioPacketReadWriter** (*read_write_closer:* `libp2p.io.abc.ReadWriteCloser`)Bases: `libp2p.io.msgio.FixedSizeLenMsgReadWriter`**size_len_bytes:** `int = 4`

```
class libp2p.security.secio.transport.SessionParameters
```

```
Bases: object
```

```
local_encryption_parameters: EncryptionParameters = None
```

```
local_peer: PeerID = None
```

```
order: int = None
```

```
remote_encryption_parameters: EncryptionParameters = None
```

```
remote_peer: PeerID = None
```

```
shared_key: bytes = None
```

```
class libp2p.security.secio.transport.Transport (local_key_pair:  

libp2p.crypto.keys.KeyPair, se-  

ecure_bytes_provider: Callable[[int],  

bytes] = <function de-  

fault_secure_bytes_provider>)
```

```
Bases: libp2p.security.base_transport.BaseSecureTransport
```

Transport provides a security upgrader for a `IRawConnection`, following the *secio* protocol defined in the libp2p specs.

```
get_nonce () → bytes
```

```
async secure_inbound (conn: libp2p.network.connection.raw_connection_interface.IRawConnection)  

→ libp2p.security.secure_conn_interface.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: libp2p.network.connection.raw_connection_interface.IRawConnection,  

peer_id: libp2p.peer.id.ID) → libp2p.security.secure_conn_interface.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:  

libp2p.peer.id.ID,  

local_private_key:  

libp2p.crypto.keys.PrivateKey,  

conn:  

libp2p.network.connection.raw_connection_i-  

remote_peer:  

libp2p.peer.id.ID  

= None) →  

libp2p.security.secure_conn_interface.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: libp2p.peer.id.ID,
                                              local_private_key:
                                              libp2p.crypto.keys.PrivateKey, re-
                                              mote_peer: libp2p.peer.id.ID,
                                              remote_permanent_pubkey:
                                              libp2p.crypto.keys.PublicKey,
                                              is_initiator: bool)
```

Bases: *libp2p.security.secure_conn_interface.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() → libp2p.peer.id.ID

get_local_private_key() → libp2p.crypto.keys.PrivateKey

get_remote_peer() → libp2p.peer.id.ID

get_remote_public_key() → Optional[libp2p.crypto.keys.PublicKey]

local_peer: ID = None

local_private_key: PrivateKey = None

remote_peer: ID = None

remote_permanent_pubkey: PublicKey = None

libp2p.security.base_transport module

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

Bases: *libp2p.security.secure_transport_interface.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: *libp2p.exceptions.BaseLibp2pError*

libp2p.security.secure_conn_interface module

class libp2p.security.secure_conn_interface.**AbstractSecureConn**

Bases: *abc.ABC*

abstract `get_local_peer()` → libp2p.peer.id.ID

abstract `get_local_private_key()` → libp2p.crypto.keys.PrivateKey

abstract `get_remote_peer()` → libp2p.peer.id.ID

abstract `get_remote_public_key()` → libp2p.crypto.keys.PublicKey

class libp2p.security.secure_conn_interface.**ISecureConn**

Bases: *libp2p.security.secure_conn_interface.AbstractSecureConn*, *libp2p.network.connection.raw_connection_interface.IRawConnection*

is_initiator = None

libp2p.security.secure_transport_interface module

class libp2p.security.secure_transport_interface.**ISecureTransport**

Bases: *abc.ABC*

abstract `async secure_inbound(conn: libp2p.network.connection.raw_connection_interface.IRawConnection)`
→ libp2p.security.secure_conn_interface.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: libp2p.network.connection.raw_connection_interface.IRawConnection, peer_id: libp2p.peer.id.ID)`
→ libp2p.security.secure_conn_interface.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: Map-ping[NewType.<locals>.new_type, libp2p.security.secure_transport_interface*)

Bases: *abc.ABC*

SSMuxer is a multistream stream security transport multiplexer.

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

add_transport (*protocol*: *NewType.<locals>.new_type*, *transport*: *libp2p.security.secure_transport_interface.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 = None**

multiselect_client: **MultiselectClient = None**

async secure_inbound (*conn*: *libp2p.network.connection.raw_connection_interface.IRawConnection*) → *libp2p.security.secure_conn_interface.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*: *libp2p.network.connection.raw_connection_interface.IRawConnection*, *peer_id*: *libp2p.peer.id.ID*) → *libp2p.security.secure_conn_interface.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*: *libp2p.network.connection.raw_connection_interface.IRawConnection*, *is_initiator*: *bool*) → *libp2p.security.secure_transport_interface.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]' = None**

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
    Bases: enum.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: tuple

    property channel_id
        Alias for field number 0

    property is_initiator
        Alias for field number 1

```

libp2p.stream_muxer.mplex.exceptions module

```

exception libp2p.stream_muxer.mplex.exceptions.MplexError
    Bases: libp2p.stream_muxer.exceptions.MuxedConnError

exception libp2p.stream_muxer.mplex.exceptions.MplexStreamClosed
    Bases: libp2p.stream_muxer.exceptions.MuxedStreamClosed

exception libp2p.stream_muxer.mplex.exceptions.MplexStreamEOF
    Bases: libp2p.stream_muxer.exceptions.MuxedStreamEOF

exception libp2p.stream_muxer.mplex.exceptions.MplexStreamReset
    Bases: libp2p.stream_muxer.exceptions.MuxedStreamReset

exception libp2p.stream_muxer.mplex.exceptions.MplexUnavailable
    Bases: libp2p.stream_muxer.exceptions.MuxedConnUnavailable

```

libp2p.stream_muxer.mplex.mplex module

```

class libp2p.stream_muxer.mplex.mplex.Mplex(secured_conn:
                                             libp2p.security.secure_conn_interface.ISecureConn,
                                             peer_id: libp2p.peer.id.ID)

    Bases: libp2p.stream_muxer.abc.IMuxedConn

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

    async accept_stream() → libp2p.stream_muxer.abc.IMuxedStream
        accepts a muxed stream opened by the other end.

    async close() → None
        close the stream muxer and underlying secured connection.

    event_closed: trio.Event = None

    event_shutting_down: trio.Event = None

    event_started: trio.Event = None

```

async handle_incoming () → None

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

property is_closed

check connection is fully closed.

Returns true if successful

property is_initiator

if this connection is the initiator.

new_stream_receive_channel: 'trio.MemoryReceiveChannel[IMuxedStream]' = None

new_stream_send_channel: 'trio.MemorySendChannel[IMuxedStream]' = None

next_channel_id: int = None

async open_stream () → libp2p.stream_muxer.abc.IMuxedStream

creates a new muxed_stream.

Returns a new MplexStream

peer_id: ID = None

async read_message () → Tuple[int, int, bytes]

Read a single message off of the secured connection.

Returns stream_id, flag, message contents

secured_conn: ISecureConn = None

async send_message (*flag:* libp2p.stream_muxer.mplex.constants.HeaderTags, *data:* Optional[bytes], *stream_id:* libp2p.stream_muxer.mplex.datastructures.StreamID) → int

sends 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 () → None

start the multiplexer.

streams: Dict[StreamID, MplexStream] = None

streams_lock: trio.Lock = None

streams_msg_channels: Dict[StreamID, 'trio.MemorySendChannel[bytes]'] = None

async write_to_stream (*_bytes:* bytes) → None

writes 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:
    libp2p.stream_muxer.mplex.datastructures.StreamID,
    muxed_conn: Mplex, incoming_data_channel:
    trio.MemoryReceiveChannel[bytes])
```

Bases: *libp2p.stream_muxer.abc.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: trio.Lock = None

event_local_closed: trio.Event = None

event_remote_closed: trio.Event = None

event_reset: trio.Event = None

incoming_data_channel: 'trio.MemoryReceiveChannel[bytes]' = None

property is_initiator

muxed_conn: 'Mplex' = None

name: str = None

async read (n: int = 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 = None

async reset () → None

closes 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 = None

async write (data: bytes) → None

write to stream.

Returns number of bytes written

```
write_deadline: int = None
```

Module contents

Submodules

libp2p.stream_muxer.abc module

```
class libp2p.stream_muxer.abc.IMuxedConn (conn: libp2p.security.secure_conn_interface.ISecureConn,  
peer_id: libp2p.peer.id.ID)
```

Bases: `abc.ABC`

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

```
abstract async accept_stream () → libp2p.stream_muxer.abc.IMuxedStream  
accepts a muxed stream opened by the other end.
```

```
abstract async close () → None  
close connection.
```

```
event_started: trio.Event = None
```

```
abstract property is_closed  
check connection is fully closed.
```

Returns true if successful

```
abstract property is_initiator  
if this connection is the initiator.
```

```
abstract async open_stream () → libp2p.stream_muxer.abc.IMuxedStream  
creates a new muxed_stream.
```

Returns a new `IMuxedStream` stream

```
peer_id: ID = None
```

```
abstract async start () → None  
start the multiplexer.
```

```
class libp2p.stream_muxer.abc.IMuxedStream
```

Bases: `libp2p.io.abc.ReadWriteCloser`

```
muxed_conn: IMuxedConn = None
```

```
abstract async reset () → None  
closes both ends of the stream tells this remote side to hang up.
```

```
abstract set_deadline (tll: int) → bool  
set deadline for muxed stream.
```

Returns a new stream

libp2p.stream_muxer.exceptions module

exception libp2p.stream_muxer.exceptions.**MuxedConnError**

Bases: *libp2p.exceptions.BaseLibp2pError*

exception libp2p.stream_muxer.exceptions.**MuxedConnUnavailable**

Bases: *libp2p.stream_muxer.exceptions.MuxedConnError*

exception libp2p.stream_muxer.exceptions.**MuxedStreamClosed**

Bases: *libp2p.stream_muxer.exceptions.MuxedStreamError*

exception libp2p.stream_muxer.exceptions.**MuxedStreamEOF**

Bases: *libp2p.stream_muxer.exceptions.MuxedStreamError*, *EOFError*

exception libp2p.stream_muxer.exceptions.**MuxedStreamError**

Bases: *libp2p.exceptions.BaseLibp2pError*

exception libp2p.stream_muxer.exceptions.**MuxedStreamReset**

Bases: *libp2p.stream_muxer.exceptions.MuxedStreamError*

libp2p.stream_muxer.muxer_multistream module

class libp2p.stream_muxer.muxer_multistream.**MuxerMultistream** (*muxer_transports_by_protocol:*
Map-
ping[NewType.<locals>.new_type,
Type[libp2p.stream_muxer.abc.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:* *NewType.<locals>.new_type,* *transport:*
Type[libp2p.stream_muxer.abc.IMuxedConn]) → 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** = None

multiselect_client: **MultiselectClient** = None

async new_conn (*conn:* *libp2p.security.secure_conn_interface.ISecureConn,* *peer_id:*
libp2p.peer.id.ID) → *libp2p.stream_muxer.abc.IMuxedConn*

async select_transport (*conn:* *libp2p.network.connection.raw_connection_interface.IRawConnection*)
→ *Type[libp2p.stream_muxer.abc.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*, *TMuxerClass*]' = None

Module contents

`libp2p.tools` package

Subpackages

`libp2p.tools.pubsub` package

Submodules

`libp2p.tools.pubsub.dummy_account_node` module

`libp2p.tools.pubsub.floodsub_integration_test_settings` module

`libp2p.tools.pubsub.utils` module

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

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

```
libp2p.tools.pubsub.utils.make_pubsub_msg (origin_id: libp2p.peer.id.ID, topic_ids: Sequence[str], data: bytes, seqno: bytes) →  
libp2p.pubsub.pb.rpc_pb2.Message
```

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

Module contents

The interop module is left out for now, because of the extra dependencies it requires.

Submodules

`libp2p.tools.constants` module

```
class libp2p.tools.constants.GossipsubParams (degree, degree_low, degree_high,  
                                             time_to_live, gossip_window, gos-  
                                             sip_history, heartbeat_initial_delay,  
                                             heartbeat_interval)
```

Bases: `tuple`

property `degree`

Alias for field number 0

property `degree_high`

Alias for field number 2

property degree_low

Alias for field number 1

property gossip_history

Alias for field number 5

property gossip_window

Alias for field number 4

property heartbeat_initial_delay

Alias for field number 6

property heartbeat_interval

Alias for field number 7

property time_to_live

Alias for field number 3

libp2p.tools.factories module**libp2p.tools.utils module**

async libp2p.tools.utils.**connect** (*node1*: libp2p.host.host_interface.IHost, *node2*: libp2p.host.host_interface.IHost) → None

Connect node1 to node2.

async libp2p.tools.utils.**connect_swarm** (*swarm_0*: libp2p.network.swarm.Swarm, *swarm_1*: libp2p.network.swarm.Swarm) → None

libp2p.tools.utils.**create_echo_stream_handler** (*ack_prefix*: str) → Callable[[libp2p.network.stream.net_stream_interface.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: libp2p.transport.transport_interface.ITransport

create_listener (*handler_function*: Callable[[libp2p.io.abc.ReadWriteCloser], Awaitable[None]]) → libp2p.transport.tcp.tcp.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.multiaddr.Multiaddr*) → libp2p.network.connection.raw_connection_interface.IRawConnectionInterface
 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[[libp2p.io.abc.ReadWriteCloser], Awaitable[None]]*)

Bases: *libp2p.transport.listener_interface.IListener*

async close () → None

get_addrs () → Tuple[multiaddr.multiaddr.Multiaddr, ...]
 retrieve list of addresses the listener is listening on.

Returns return list of addrs

async listen (*maddr: multiaddr.multiaddr.Multiaddr, nursery: trio.Nursery*) → None
 put listener in listening mode and wait for incoming connections.

Parameters **maddr** – maddr of peer

Returns return True if successful

listeners: List[trio.SocketListener] = None

Module contents

Submodules

libp2p.transport.exceptions module

exception libp2p.transport.exceptions.**MuxerUpgradeFailure**
 Bases: *libp2p.transport.exceptions.UpgradeFailure*

exception libp2p.transport.exceptions.**OpenConnectionError**
 Bases: *libp2p.exceptions.BaseLibp2pError*

exception libp2p.transport.exceptions.**SecurityUpgradeFailure**
 Bases: *libp2p.transport.exceptions.UpgradeFailure*

exception libp2p.transport.exceptions.**UpgradeFailure**
 Bases: *libp2p.exceptions.BaseLibp2pError*

libp2p.transport.listener_interface module

class libp2p.transport.listener_interface.**IListener**
 Bases: *abc.ABC*

abstract async close () → None

abstract get_addrs () → Tuple[multiaddr.multiaddr.Multiaddr, ...]
 retrieve list of addresses the listener is listening on.

Returns return list of addrs

abstract async listen (*maddr: multiaddr:multiaddr.Multiaddr, nursery: trio.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.ABC

abstract create_listener (*handler_function: Callable[[libp2p.io.abc.ReadWriteCloser], Awaitable[None]]*) → libp2p.transport.listener_interface.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:multiaddr.Multiaddr*) → libp2p.network.connection.raw_connection_interface.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: Map-ping[NewType.<locals>.new_type, libp2p.security.secure_transport_interface.ISecureTransport], muxer_transports_by_protocol: Map-ping[NewType.<locals>.new_type, Type[libp2p.stream_muxer.abc.IMuxedConn]]*)

Bases: object

muxer_multistream: MuxerMultistream = None

security_multistream: SecurityMultistream = None

async upgrade_connection (*conn: libp2p.security.secure_conn_interface.ISecureConn, peer_id: libp2p.peer.id.ID*) → libp2p.stream_muxer.abc.IMuxedConn
 Upgrade secured connection to a muxed connection.

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

```
async upgrade_security (raw_conn: libp2p.network.connection.raw_connection_interface.IRawConnection,  
                        peer_id: libp2p.peer.id.ID, is_initiator: bool) →  
                        libp2p.security.secure_conn_interface.ISecureConn  
Upgrade conn to a secured connection.
```

Module contents

1.1.2 Submodules

1.1.3 libp2p.exceptions module

exception libp2p.exceptions.**BaseLibp2pError**
Bases: *Exception*

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

exception libp2p.exceptions.**ParseError**
Bases: *libp2p.exceptions.BaseLibp2pError*

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

1.1.4 libp2p.typing module

1.1.5 libp2p.utils module

async libp2p.utils.**decode_uvarint_from_stream** (reader: *libp2p.io.abc.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_prefixed** (msg_bytes: *bytes*) → bytes

async libp2p.utils.**read_delim** (reader: *libp2p.io.abc.Reader*) → bytes

async libp2p.utils.**read_varint_prefixed_bytes** (reader: *libp2p.io.abc.Reader*) → bytes

1.1.6 Module contents

libp2p.**generate_new_rsa_identity** () → libp2p.crypto.keys.KeyPair

libp2p.**generate_peer_id_from** (key_pair: *libp2p.crypto.keys.KeyPair*) → libp2p.peer.id.ID

```
libp2p.new_host(key_pair: libp2p.crypto.keys.KeyPair = None, muxer_opt: Mapping[NewType.<locals>.new_type, Type[libp2p.stream_muxer.abc.IMuxedConn]] = None, sec_opt: Mapping[NewType.<locals>.new_type, libp2p.security.secure_transport_interface.ISecureTransport] = None, peerstore_opt: libp2p.peer.peerstore_interface.IPeerStore = None, disc_opt: libp2p.routing.interfaces.IPeerRouting = None) → libp2p.host.host_interface.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: libp2p.crypto.keys.KeyPair = None, muxer_opt: Mapping[NewType.<locals>.new_type, Type[libp2p.stream_muxer.abc.IMuxedConn]] = None, sec_opt: Mapping[NewType.<locals>.new_type, libp2p.security.secure_transport_interface.ISecureTransport] = None, peerstore_opt: libp2p.peer.peerstore_interface.IPeerStore = None) → libp2p.network.network_interface.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

1.2 Release Notes

1.2.1 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 Kademia 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)

1.2.2 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)

1.2.3 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

1.2.4 v0.1.2

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

1.3 examples package

1.3.1 Subpackages

`examples.chat` package

Submodules

`examples.chat.chat` module

`examples.chat.chat.main()` → None

async `examples.chat.chat.read_data` (*stream: libp2p.network.stream.net_stream_interface.INetStream*)
→ None

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

async `examples.chat.chat.write_data` (*stream: libp2p.network.stream.net_stream_interface.INetStream*)
→ None

Module contents

1.3.2 Module contents

INDICES AND TABLES

- genindex
- modindex

PYTHON MODULE INDEX

e

examples, 73

examples.chat, 73

examples.chat.chat, 73

l

libp2p, 70

libp2p.crypto, 9

libp2p.crypto.authenticated_encryption,
4

libp2p.crypto.ecc, 4

libp2p.crypto.ed25519, 5

libp2p.crypto.exceptions, 6

libp2p.crypto.key_exchange, 6

libp2p.crypto.keys, 6

libp2p.crypto.pb, 4

libp2p.crypto.pb.crypto_pb2, 3

libp2p.crypto.rsa, 7

libp2p.crypto.secp256k1, 8

libp2p.crypto.serialization, 8

libp2p.exceptions, 70

libp2p.host, 12

libp2p.host.basic_host, 9

libp2p.host.defaults, 10

libp2p.host.exceptions, 10

libp2p.host.host_interface, 10

libp2p.host.ping, 11

libp2p.host.routed_host, 12

libp2p.identity, 13

libp2p.identity.identify, 13

libp2p.identity.identify.pb, 13

libp2p.identity.identify.pb.identify_pb2,
12

libp2p.identity.identify.protocol, 13

libp2p.io, 15

libp2p.io.abc, 13

libp2p.io.exceptions, 14

libp2p.io.msgio, 14

libp2p.io.utils, 15

libp2p.network, 22

libp2p.network.connection, 17

libp2p.network.connection.exceptions,
15

libp2p.network.connection.net_connection_interface,
16

libp2p.network.connection.raw_connection,
16

libp2p.network.connection.raw_connection_interface,
16

libp2p.network.connection.swarm_connection,
16

libp2p.network.exceptions, 18

libp2p.network.network_interface, 18

libp2p.network.notifee_interface, 19

libp2p.network.stream, 18

libp2p.network.stream.exceptions, 17

libp2p.network.stream.net_stream, 17

libp2p.network.stream.net_stream_interface,
18

libp2p.network.swarm, 20

libp2p.peer, 31

libp2p.peer.addrbook_interface, 22

libp2p.peer.id, 23

libp2p.peer.peerdata, 23

libp2p.peer.peerdata_interface, 25

libp2p.peer.peerinfo, 26

libp2p.peer.peermetadata_interface, 26

libp2p.peer.peerstore, 26

libp2p.peer.peerstore_interface, 29

libp2p.protocol_muxer, 35

libp2p.protocol_muxer.exceptions, 31

libp2p.protocol_muxer.multiselect, 31

libp2p.protocol_muxer.multiselect_client,
32

libp2p.protocol_muxer.multiselect_client_interface,
33

libp2p.protocol_muxer.multiselect_communicator,
34

libp2p.protocol_muxer.multiselect_communicator_int
34

libp2p.protocol_muxer.multiselect_muxer_interface,
34

libp2p.pubsub, 47

- libp2p.pubsub.abc, 38
- libp2p.pubsub.exceptions, 39
- libp2p.pubsub.floodsub, 39
- libp2p.pubsub.gossipsub, 40
- libp2p.pubsub.mcache, 42
- libp2p.pubsub.pb, 38
- libp2p.pubsub.pb.rpc_pb2, 35
- libp2p.pubsub.pubsub, 43
- libp2p.pubsub.pubsub_notiffee, 46
- libp2p.pubsub.subscription, 47
- libp2p.pubsub.validators, 47
- libp2p.routing, 48
- libp2p.routing.interfaces, 47
- libp2p.security, 60
 - libp2p.security.base_session, 58
 - libp2p.security.base_transport, 58
 - libp2p.security.exceptions, 59
 - libp2p.security.insecure, 50
 - libp2p.security.insecure.pb, 49
 - libp2p.security.insecure.pb.plaintext_pb2, 48
 - libp2p.security.insecure.transport, 49
 - libp2p.security.noise, 54
 - libp2p.security.noise.exceptions, 51
 - libp2p.security.noise.io, 51
 - libp2p.security.noise.messages, 52
 - libp2p.security.noise.patterns, 53
 - libp2p.security.noise.pb, 51
 - libp2p.security.noise.pb.noise_pb2, 50
 - libp2p.security.noise.transport, 54
 - libp2p.security.secio, 58
 - libp2p.security.secio.exceptions, 55
 - libp2p.security.secio.pb, 55
 - libp2p.security.secio.pb.spipe_pb2, 54
 - libp2p.security.secio.transport, 56
 - libp2p.security.secure_conn_interface, 59
 - libp2p.security.secure_transport_interface, 59
 - libp2p.security.security_multistream, 59
- libp2p.stream_muxer, 66
 - libp2p.stream_muxer.abc, 64
 - libp2p.stream_muxer.exceptions, 65
 - libp2p.stream_muxer.mplex, 64
 - libp2p.stream_muxer.mplex.constants, 60
 - libp2p.stream_muxer.mplex.datastructures, 61
 - libp2p.stream_muxer.mplex.exceptions, 61
 - libp2p.stream_muxer.mplex.mplex, 61
 - libp2p.stream_muxer.mplex.mplex_stream, 63
 - libp2p.stream_muxer.mplex_multistream, 65
 - libp2p.tools, 67
 - libp2p.tools.constants, 66
 - libp2p.tools.pubsub, 66
 - libp2p.tools.pubsub.utils, 66
 - libp2p.tools.utils, 67
 - libp2p.transport, 70
 - libp2p.transport.exceptions, 68
 - libp2p.transport.listener_interface, 68
 - libp2p.transport.tcp, 68
 - libp2p.transport.tcp.tcp, 67
 - libp2p.transport.transport_interface, 69
 - libp2p.transport.typing, 69
 - libp2p.transport.upgrader, 69
 - libp2p.typing, 70
 - libp2p.utils, 70

INDEX

A

- `AbstractSecureConn` (class in `libp2p.security.secure_conn_interface`), 59
- `accept_stream()` (`libp2p.stream_muxer.abc.IMuxedConn` method), 64
- `accept_stream()` (`libp2p.stream_muxer.mplex.mplex.Mplex` method), 61
- `add_addr()` (`libp2p.peer.addrbook_interface.IAddrBook` method), 22
- `add_addr()` (`libp2p.peer.peerstore.PeerStore` method), 26
- `add_addr()` (`libp2p.peer.peerstore_interface.IPeerStore` method), 29
- `add_addrs()` (`libp2p.peer.addrbook_interface.IAddrBook` method), 22
- `add_addrs()` (`libp2p.peer.peerdata.PeerData` method), 23
- `add_addrs()` (`libp2p.peer.peerdata_interface.IPeerData` method), 25
- `add_addrs()` (`libp2p.peer.peerstore.PeerStore` method), 26
- `add_addrs()` (`libp2p.peer.peerstore_interface.IPeerStore` method), 29
- `add_conn()` (`libp2p.network.swarm.Swarm` method), 20
- `add_handler()` (`libp2p.protocol_muxer.multiselect.Multiselect` method), 31
- `add_handler()` (`libp2p.protocol_muxer.multiselect_muxer.MultiselectMuxer` method), 34
- `add_key_pair()` (`libp2p.peer.peerstore.PeerStore` method), 27
- `add_key_pair()` (`libp2p.peer.peerstore_interface.IPeerStore` method), 29
- `add_peer()` (`libp2p.pubsub.abc.IPubsubRouter` method), 38
- `add_peer()` (`libp2p.pubsub.floodsub.FloodSub` method), 39
- `add_peer()` (`libp2p.pubsub.gossipsub.GossipSub` method), 40
- `add_privkey()` (`libp2p.peer.peerdata.PeerData` method), 23
- `add_privkey()` (`libp2p.peer.peerdata_interface.IPeerData` method), 25
- `add_privkey()` (`libp2p.peer.peerstore.PeerStore` method), 27
- `add_privkey()` (`libp2p.peer.peerstore_interface.IPeerStore` method), 29
- `add_protocols()` (`libp2p.peer.peerdata.PeerData` method), 23
- `add_protocols()` (`libp2p.peer.peerdata_interface.IPeerData` method), 25
- `add_protocols()` (`libp2p.peer.peerstore.PeerStore` method), 27
- `add_protocols()` (`libp2p.peer.peerstore_interface.IPeerStore` method), 29
- `add_pubkey()` (`libp2p.peer.peerdata.PeerData` method), 23
- `add_pubkey()` (`libp2p.peer.peerdata_interface.IPeerData` method), 25
- `add_pubkey()` (`libp2p.peer.peerstore.PeerStore` method), 27
- `add_pubkey()` (`libp2p.peer.peerstore_interface.IPeerStore` method), 29
- `add_transport()` (`libp2p.security.security_multistream.SecurityMultistream` method), 59
- `add_transport()` (`libp2p.stream_muxer_muxer_multistream.MuxerMultistream` method), 65
- `addrs` (`libp2p.peer.peerdata.PeerData` attribute), 24
- `addrs` (`libp2p.peer.peerinfo.PeerInfo` attribute), 26
- `addrs()` (`libp2p.peer.addrbook_interface.IAddrBook` method), 23
- `addrs()` (`libp2p.peer.peerstore.PeerStore` method), 27
- `addrs()` (`libp2p.peer.peerstore_interface.IPeerStore` method), 29
- `agent_version` (`libp2p.identity.identify.pb.identify_pb2.Identify` attribute), 12
- `attach()` (`libp2p.pubsub.abc.IPubsubRouter` method), 38
- `attach()` (`libp2p.pubsub.floodsub.FloodSub` method), 39
- `attach()` (`libp2p.pubsub.gossipsub.GossipSub` method), 40
- `auth` (`libp2p.pubsub.pb.rpc_pb2.TopicDescriptor` attribute), 37

authenticate() (*libp2p.crypto.authenticated_encryption.EncryptionParameters* attribute), 4

AuthMode (*libp2p.pubsub.pb.rpc_pb2.TopicDescriptor.AuthOptions* attribute), 37

B

BaseLibp2pError, 70

BaseMsgReadWriter (*class in libp2p.io.msgio*), 14

BaseNoiseMsgReadWriter (*class in libp2p.security.noise.io*), 51

BasePattern (*class in libp2p.security.noise.patterns*), 53

BaseSecureTransport (*class in libp2p.security.base_transport*), 58

BaseSession (*class in libp2p.security.base_session*), 58

BaseSubscriptionAPI (*class in libp2p.pubsub.subscription*), 47

BasicHost (*class in libp2p.host.basic_host*), 9

C

CacheEntry (*class in libp2p.pubsub.mcache*), 42

calculate_peer_id() (*libp2p.security.secio.transport.Proposal* method), 56

channel_id() (*libp2p.stream_muxer.mplex.datastructures.StreamID* property), 61

cipher_key (*libp2p.crypto.authenticated_encryption.EncryptionParameters* attribute), 4

cipher_type (*libp2p.crypto.authenticated_encryption.EncryptionParameters* attribute), 4

cipher_type (*libp2p.security.secio.transport.EncryptionParameters* attribute), 56

ciphers (*libp2p.security.secio.pb.spice_pb2.Propose* attribute), 55

ciphers (*libp2p.security.secio.transport.Proposal* attribute), 56

clear_addrs() (*libp2p.peer.addrbook_interface.IAddrBook* method), 23

clear_addrs() (*libp2p.peer.peerdata.PeerData* method), 24

clear_addrs() (*libp2p.peer.peerdata_interface.IPeerData* method), 25

clear_addrs() (*libp2p.peer.peerstore.PeerStore* method), 27

clear_addrs() (*libp2p.peer.peerstore_interface.IPeerStore* method), 30

close() (*libp2p.host.basic_host.BasicHost* method), 9

close() (*libp2p.host.host_interface.IHost* method), 10

close() (*libp2p.io.abc.Closer* method), 13

close() (*libp2p.io.msgio.BaseMsgReadWriter* method), 14

close() (*libp2p.network.connection.raw_connection.RawConnection* method), 16

close() (*libp2p.network.connection.swarm_connection.SwarmConn* method), 16

close() (*libp2p.network.network_interface.INetwork* method), 18

close() (*libp2p.network.stream.net_stream.NetStream* method), 17

close() (*libp2p.network.swarm.Swarm* method), 20

close() (*libp2p.security.insecure.transport.InsecureSession* method), 49

close() (*libp2p.security.noise.io.BaseNoiseMsgReadWriter* method), 51

close() (*libp2p.security.secio.transport.SecioMsgReadWriter* method), 56

close() (*libp2p.stream_muxer.abc.IMuxedConn* method), 64

close() (*libp2p.stream_muxer.mplex.mplex.Mplex* method), 61

close() (*libp2p.stream_muxer.mplex.mplex_stream.MplexStream* method), 63

close() (*libp2p.transport.listener_interface.IListener* method), 68

close() (*libp2p.transport.tcp.tcp.TCPListener* method), 68

close_lock (*libp2p.stream_muxer.mplex.mplex_stream.MplexStream* attribute), 63

close_peer() (*libp2p.network.network_interface.INetwork* method), 18

close_peer() (*libp2p.network.swarm.Swarm* method), 20

closed_stream() (*libp2p.network.notifee_interface.INotifee* method), 19

closed_stream() (*libp2p.pubsub.pubsub_notifee.PubsubNotifee* method), 46

CloseInitiator (*libp2p.stream_muxer.mplex.constants.HeaderTags* attribute), 60

Closer (*class in libp2p.io.abc*), 13

CloseReceiver (*libp2p.stream_muxer.mplex.constants.HeaderTags* attribute), 60

common_stream_handler (*libp2p.network.swarm.Swarm* attribute), 20

connect() (*in module libp2p.tools.utils*), 67

connect() (*libp2p.host.basic_host.BasicHost* method), 9

connect() (*libp2p.host.host_interface.IHost* method), 10

connect() (*libp2p.host.routed_host.RoutedHost* method), 12

connect_some() (*in module libp2p.tools.pubsub.utils*), 66

connect_swarm() (*in module libp2p.tools.utils*), 67

connected() (*libp2p.network.notifee_interface.INotifee* method), 19

connected() (*libp2p.pubsub.pubsub_notifee.PubsubNotifee* method), 19

- method*), 46
- ConnectionFailure, 10
- connections (*libp2p.network.network_interface.INetworkInterface* attribute), 18
- connections (*libp2p.network.network_interface.INetworkService* attribute), 19
- connections (*libp2p.network.swarm.Swarm* attribute), 20
- continuously_read_stream() (*libp2p.pubsub.pubsub.Pubsub* method), 43
- control (*libp2p.pubsub.pb.rpc_pb2.RPC* attribute), 36
- ControlGraft (class in *libp2p.pubsub.pb.rpc_pb2*), 35
- ControlIHave (class in *libp2p.pubsub.pb.rpc_pb2*), 35
- ControlIWant (class in *libp2p.pubsub.pb.rpc_pb2*), 35
- ControlMessage (class in *libp2p.pubsub.pb.rpc_pb2*), 35
- ControlPrune (class in *libp2p.pubsub.pb.rpc_pb2*), 36
- counter (*libp2p.pubsub.pubsub.Pubsub* attribute), 43
- create_default_stream_handler() (in module *libp2p.network.swarm*), 22
- create_echo_stream_handler() (in module *libp2p.tools.utils*), 67
- create_ephemeral_key_pair() (in module *libp2p.crypto.key_exchange*), 6
- create_listener() (*libp2p.transport.tcp.tcp.TCP* method), 67
- create_listener() (*libp2p.transport.transport_interface.ITransport* method), 69
- create_new_key_pair() (in module *libp2p.crypto.ecc*), 5
- create_new_key_pair() (in module *libp2p.crypto.ed25519*), 5
- create_new_key_pair() (in module *libp2p.crypto.rsa*), 7
- create_new_key_pair() (in module *libp2p.crypto.secp256k1*), 8
- create_noise_state() (*libp2p.security.noise.patterns.BasePattern* method), 53
- create_secure_session() (in module *libp2p.security.secio.transport*), 57
- CryptographyError, 6
- curve_type (*libp2p.security.secio.transport.EncryptionParameters* attribute), 56
- data (*libp2p.crypto.pb.crypto_pb2.PrivateKey* attribute), 3
- data (*libp2p.crypto.pb.crypto_pb2.PublicKey* attribute), 3
- data (*libp2p.pubsub.pb.rpc_pb2.Message* attribute), 36
- data (*libp2p.security.noise.pb.noise_pb2.NoiseHandshakePayload* attribute), 50
- dead_peer_receive_channel (*libp2p.pubsub.pubsub.Pubsub* attribute), 43
- dead_peers_queue (*libp2p.pubsub.pubsub_notifier.PubsubNotifier* attribute), 46
- decode_uvarint_from_stream() (in module *libp2p.utils*), 70
- decrypt() (*libp2p.io.abc.Encrypter* method), 13
- decrypt() (*libp2p.security.noise.io.NoiseHandshakeReadWriter* method), 51
- decrypt() (*libp2p.security.noise.io.NoiseTransportReadWriter* method), 52
- decrypt() (*libp2p.security.secio.transport.SecioMsgReadWriter* method), 56
- decrypt_if_valid() (*libp2p.crypto.authenticated_encryption.MacAndCipher* method), 4
- DecryptionFailedException, 14
- default_secure_bytes_provider() (in module *libp2p.security.base_transport*), 58
- degree (*libp2p.pubsub.gossipsub.GossipSub* attribute), 40
- degree() (*libp2p.tools.constants.GossipsubParams* property), 66
- degree_high (*libp2p.pubsub.gossipsub.GossipSub* attribute), 40
- degree_high() (*libp2p.tools.constants.GossipsubParams* property), 66
- degree_low (*libp2p.pubsub.gossipsub.GossipSub* attribute), 40
- degree_low() (*libp2p.tools.constants.GossipsubParams* property), 66
- dense_connect() (in module *libp2p.tools.pubsub.utils*), 66
- DESCRIPTOR (*libp2p.crypto.pb.crypto_pb2.PrivateKey* attribute), 3
- DESCRIPTOR (*libp2p.crypto.pb.crypto_pb2.PublicKey* attribute), 3
- DESCRIPTOR (*libp2p.identity.identify.pb.identify_pb2.Identify* attribute), 12
- DESCRIPTOR (*libp2p.pubsub.pb.rpc_pb2.ControlGraft* attribute), 35
- DESCRIPTOR (*libp2p.pubsub.pb.rpc_pb2.ControlIHave* attribute), 35
- DESCRIPTOR (*libp2p.pubsub.pb.rpc_pb2.ControlIWant* attribute), 35
- DESCRIPTOR (*libp2p.pubsub.pb.rpc_pb2.ControlMessage* attribute), 35
- DESCRIPTOR (*libp2p.pubsub.pb.rpc_pb2.ControlPrune* attribute), 35

D

attribute), 36
 DESCRIPTOR (*libp2p.pubsub.pb.rpc_pb2.Message attribute*), 36
 DESCRIPTOR (*libp2p.pubsub.pb.rpc_pb2.RPC attribute*), 36
 DESCRIPTOR (*libp2p.pubsub.pb.rpc_pb2.RPC.SubOpts attribute*), 36
 DESCRIPTOR (*libp2p.pubsub.pb.rpc_pb2.TopicDescriptor attribute*), 37
 DESCRIPTOR (*libp2p.pubsub.pb.rpc_pb2.TopicDescriptor.AuthOpts attribute*), 37
 DESCRIPTOR (*libp2p.pubsub.pb.rpc_pb2.TopicDescriptor.EncOpts attribute*), 37
 DESCRIPTOR (*libp2p.security.insecure.pb.plaintext_pb2.Exchange attribute*), 48
 DESCRIPTOR (*libp2p.security.noise.pb.noise_pb2.NoiseHandshakePayload attribute*), 50
 DESCRIPTOR (*libp2p.security.secio.pb.spipe_pb2.Exchange attribute*), 54
 DESCRIPTOR (*libp2p.security.secio.pb.spipe_pb2.Propose attribute*), 55
 deserialize() (*libp2p.crypto.secp256k1.Secp256k1PrivateKey class method*), 8
 deserialize() (*libp2p.crypto.secp256k1.Secp256k1PublicKey class method*), 8
 deserialize() (*libp2p.security.noise.messages.NoiseHandshakePayload class method*), 52
 deserialize() (*libp2p.security.secio.transport.Proposal class method*), 56
 deserialize_from_protobuf() (*libp2p.crypto.keys.PrivateKey class method*), 6
 deserialize_from_protobuf() (*libp2p.crypto.keys.PublicKey class method*), 7
 deserialize_private_key() (*in module libp2p.crypto.serialization*), 8
 deserialize_public_key() (*in module libp2p.crypto.serialization*), 8
 dial() (*libp2p.transport.tcp.tcp.TCP method*), 67
 dial() (*libp2p.transport.transport_interface.ITransport method*), 69
 dial_addr() (*libp2p.network.swarm.Swarm method*), 20
 dial_peer() (*libp2p.network.network_interface.INetwork method*), 18
 dial_peer() (*libp2p.network.swarm.Swarm method*), 21
 digest() (*libp2p.peer.id.IdentityHash method*), 23
 disconnect() (*libp2p.host.basic_host.BasicHost method*), 9
 disconnect() (*libp2p.host.host_interface.IHost method*), 10
 disconnected() (*libp2p.network.notifee_interface.INotifee method*), 19
 disconnected() (*libp2p.pubsub.pubsub_notifee.PubsubNotifee method*), 46
E
 early_data (*libp2p.security.noise.messages.NoiseHandshakePayload attribute*), 52
 early_data (*libp2p.security.noise.patterns.BasePattern attribute*), 53
 early_data (*libp2p.security.noise.patterns.PatternXX attribute*), 53
 early_data (*libp2p.security.noise.transport.Transport attribute*), 54
 ECC_P256 (*libp2p.crypto.keys.KeyType attribute*), 6
 Ed25519PrivateKey (*class in libp2p.crypto.ed25519*), 4
 ECCPublicKey (*class in libp2p.crypto.ed25519*), 5
 Ed25519PrivateKey (*libp2p.crypto.keys.KeyType attribute*), 6
 Ed25519PrivateKey (*class in libp2p.crypto.ed25519*), 5
 Ed25519PublicKey (*class in libp2p.crypto.ed25519*), 5
 emit_control_message() (*libp2p.pubsub.gossipsub.GossipSub method*), 40
 emit_graft() (*libp2p.pubsub.gossipsub.GossipSub method*), 40
 emit_ihave() (*libp2p.pubsub.gossipsub.GossipSub method*), 41
 emit_iwant() (*libp2p.pubsub.gossipsub.GossipSub method*), 41
 emit_prune() (*libp2p.pubsub.gossipsub.GossipSub method*), 41
 enc (*libp2p.pubsub.pb.rpc_pb2.TopicDescriptor attribute*), 37
 EncMode (*libp2p.pubsub.pb.rpc_pb2.TopicDescriptor.EncOpts attribute*), 37
 encode_delim() (*in module libp2p.utils*), 70
 encode_msg() (*libp2p.io.msgio.BaseMsgReadWriter method*), 14
 encode_msg() (*libp2p.io.msgio.FixedSizeLenMsgReadWriter method*), 15
 encode_msg() (*libp2p.io.msgio.VarIntLengthMsgReadWriter method*), 15
 encode_msg_with_length() (*in module libp2p.io.msgio*), 15
 encode_uvarint() (*in module libp2p.utils*), 70
 encode_varint_prefixed() (*in module libp2p.utils*), 70
 encrypt() (*libp2p.crypto.authenticated_encryption.MacAndCipher method*), 4
 encrypt() (*libp2p.io.abc.Encrypter method*), 13
 encrypt() (*libp2p.security.noise.io.NoiseHandshakeReadWriter method*), 51

encrypt () (*libp2p.security.noise.io.NoiseTransportReadWriter* method), 52
 encrypt () (*libp2p.security.secio.transport.SecioMsgReadWriter* method), 56
 EncryptedMsgReadWriter (*class in libp2p.io.abc*), 13
 Encrypter (*class in libp2p.io.abc*), 13
 EncryptionParameters (*class in libp2p.crypto.authenticated_encryption*), 4
 EncryptionParameters (*class in libp2p.security.secio.transport*), 56
 ephemeral_public_key (*libp2p.security.secio.pb.spipe_pb2.Exchange* attribute), 54
 ephemeral_public_key (*libp2p.security.secio.transport.EncryptionParameters* attribute), 56
 event_closed (*libp2p.network.connection.swarm_connection.SwarmConn* attribute), 16
 event_closed (*libp2p.stream_muxer.mplex.mplex.MplexStream* attribute), 61
 event_handle_dead_peer_queue_started (*libp2p.pubsub.pubsub.Pubsub* attribute), 43
 event_handle_peer_queue_started (*libp2p.pubsub.pubsub.Pubsub* attribute), 43
 event_listener_nursery_created (*libp2p.network.swarm.Swarm* attribute), 21
 event_local_closed (*libp2p.stream_muxer.mplex.mplex_stream.MplexStream* attribute), 63
 event_remote_closed (*libp2p.stream_muxer.mplex.mplex_stream.MplexStream* attribute), 63
 event_reset (*libp2p.stream_muxer.mplex.mplex_stream.MplexStream* attribute), 63
 event_shutting_down (*libp2p.stream_muxer.mplex.mplex.MplexStream* attribute), 61
 event_started (*libp2p.network.connection.net_connection_interface.INetConn* attribute), 16
 event_started (*libp2p.stream_muxer.abc.IMuxedConn* attribute), 64
 event_started (*libp2p.stream_muxer.mplex.mplex.MplexStream* attribute), 61
 examples (*module*), 73
 examples.chat (*module*), 73
 examples.chat.chat (*module*), 73
 Exchange (*class in libp2p.security.insecure.pb.plaintext_pb2*), 48
 Exchange (*class in libp2p.security.secio.pb.spipe_pb2*), 54
 exchanges (*libp2p.security.secio.pb.spipe_pb2.Propose* attribute), 55
 exchanges (*libp2p.security.secio.transport.Proposal* attribute), 56
F
 fanout (*libp2p.pubsub.gossipsub.GossipSub* attribute), 41
 fanout_heartbeat () (*libp2p.pubsub.gossipsub.GossipSub* method), 41
 find_peer () (*libp2p.routing.interfaces.IPeerRouting* method), 48
 find_provider_iter () (*libp2p.routing.interfaces.IContentRouting* method), 47
 fixed_size_len_msg_read_writer (*class in libp2p.io.msgio*), 15
 from_base58 () (*libp2p.peer.id.ID* class method), 23
 from_bytes () (*libp2p.crypto.ecc.ECCPublicKey* class method), 5
 from_bytes () (*libp2p.crypto.ed25519.Ed25519PrivateKey* class method), 5
 from_bytes () (*libp2p.crypto.ed25519.Ed25519PublicKey* class method), 5
 from_bytes () (*libp2p.crypto.rsa.RSAPublicKey* class method), 7
 from_bytes () (*libp2p.crypto.secp256k1.Secp256k1PrivateKey* class method), 8
 from_bytes () (*libp2p.crypto.secp256k1.Secp256k1PublicKey* class method), 8
 from_id (*libp2p.pubsub.pb.rpc_pb2.Message* attribute), 36
 from_pubkey () (*libp2p.peer.id.ID* class method), 23
 generate_new_rsa_identity () (*in module libp2p*), 70
 generate_peer_id_from () (*in module libp2p*), 70
 get () (*libp2p.peer.peermetadata_interface.IPeerMetadata* method), 28
 get () (*libp2p.peer.peerstore.PeerStore* method), 27
 get () (*libp2p.peer.peerstore_interface.IPeerStore* method), 30
 get () (*libp2p.pubsub.abc.ISubscriptionAPI* method), 39
 get () (*libp2p.pubsub.mcache.MessageCache* method), 42
 get () (*libp2p.pubsub.subscription.TrioSubscriptionAPI* method), 47
 get_addrs () (*libp2p.host.basic_host.BasicHost* method), 9

get_addrs() (*libp2p.host.host_interface.IHost method*), 10
get_addrs() (*libp2p.peer.peerdata.PeerData method*), 24
get_addrs() (*libp2p.peer.peerdata_interface.IPeerData method*), 25
get_addrs() (*libp2p.transport.listener_interface.IListener method*), 68
get_addrs() (*libp2p.transport.tcp.tcp.TCPListener method*), 68
get_content_addressed_msg_id() (in module *libp2p.pubsub.pubsub*), 46
get_default_protocols() (in module *libp2p.host.defaults*), 10
get_hello_packet() (*libp2p.pubsub.pubsub.Pubsub method*), 43
get_id() (*libp2p.host.basic_host.BasicHost method*), 9
get_id() (*libp2p.host.host_interface.IHost method*), 11
get_local_peer() (*libp2p.security.base_session.BaseSession method*), 58
get_local_peer() (*libp2p.security.secure_conn_interface.AbstractSecureConn method*), 59
get_local_private_key() (*libp2p.security.base_session.BaseSession method*), 58
get_local_private_key() (*libp2p.security.secure_conn_interface.AbstractSecureConn method*), 59
get_metadata() (*libp2p.peer.peerdata.PeerData method*), 24
get_metadata() (*libp2p.peer.peerdata_interface.IPeerData method*), 25
get_msg_validators() (*libp2p.pubsub.pubsub.Pubsub method*), 43
get_mux() (*libp2p.host.basic_host.BasicHost method*), 9
get_mux() (*libp2p.host.host_interface.IHost method*), 11
get_network() (*libp2p.host.basic_host.BasicHost method*), 9
get_network() (*libp2p.host.host_interface.IHost method*), 11
get_nonce() (*libp2p.security.secio.transport.Transport method*), 57
get_pattern() (*libp2p.security.noise.transport.Transport method*), 54
get_peer_and_seqno_msg_id() (in module *libp2p.pubsub.pubsub*), 46
get_peer_id() (*libp2p.network.network_interface.INetwork method*), 18
get_peer_id() (*libp2p.network.swarm.Swarm method*), 21
get_peerstore() (*libp2p.host.basic_host.BasicHost method*), 9
get_private_key() (*libp2p.host.basic_host.BasicHost method*), 9
get_private_key() (*libp2p.host.host_interface.IHost method*), 11
get_privkey() (*libp2p.peer.peerdata.PeerData method*), 24
get_privkey() (*libp2p.peer.peerdata_interface.IPeerData method*), 25
get_protocol() (*libp2p.network.stream.net_stream.NetStream method*), 17
get_protocol() (*libp2p.network.stream.net_stream_interface.INetStream method*), 18
get_protocols() (*libp2p.peer.peerdata.PeerData method*), 24
get_protocols() (*libp2p.peer.peerdata_interface.IPeerData method*), 25
get_protocols() (*libp2p.peer.peerstore.PeerStore method*), 27
get_protocols() (*libp2p.peer.peerstore_interface.IPeerStore method*), 30
get_protocols() (*libp2p.protocol_muxer.multiselect_muxer_interface.MultiselectMuxer method*), 34
get_protocols() (*libp2p.pubsub.abc.IPubsubRouter method*), 38
get_protocols() (*libp2p.pubsub.floodsub.FloodSub method*), 39
get_protocols() (*libp2p.pubsub.gossipsub.GossipSub method*), 41
get_pubkey() (*libp2p.peer.peerdata.PeerData method*), 24
get_pubkey() (*libp2p.peer.peerdata_interface.IPeerData method*), 25
get_public_key() (*libp2p.crypto.ecc.ECCPrivateKey method*), 4
get_public_key() (*libp2p.crypto.ed25519.Ed25519PrivateKey method*), 5
get_public_key() (*libp2p.crypto.keys.PrivateKey method*), 6
get_public_key() (*libp2p.crypto.rsa.RSAPrivateKey method*), 7
get_public_key() (*libp2p.crypto.secp256k1.Secp256k1PrivateKey method*), 8
get_public_key() (*libp2p.host.basic_host.BasicHost method*), 9
get_public_key() (*libp2p.host.host_interface.IHost method*), 11
get_remote_peer() (*libp2p.security.base_session.BaseSession method*), 58

get_remote_peer() (*libp2p.security.secure_conn_interface.AbstractSecureConn* method), 59
 get_remote_public_key() (*libp2p.security.base_session.BaseSession* method), 58
 get_remote_public_key() (*libp2p.security.secure_conn_interface.AbstractSecureConn* method), 59
 get_streams() (*libp2p.network.connection.net_connection_interface.NetConn* method), 16
 get_streams() (*libp2p.network.connection.swarm_connection.SwarmConn* method), 16
 get_type() (*libp2p.crypto.ecc.ECCPrivateKey* method), 4
 get_type() (*libp2p.crypto.ecc.ECCPublicKey* method), 5
 get_type() (*libp2p.crypto.ed25519.Ed25519PrivateKey* method), 5
 get_type() (*libp2p.crypto.ed25519.Ed25519PublicKey* method), 5
 get_type() (*libp2p.crypto.keys.Key* method), 6
 get_type() (*libp2p.crypto.rsa.RSAPrivateKey* method), 7
 get_type() (*libp2p.crypto.rsa.RSAPublicKey* method), 7
 get_type() (*libp2p.crypto.secp256k1.Secp256k1PrivateKey* method), 8
 get_type() (*libp2p.crypto.secp256k1.Secp256k1PublicKey* method), 8
 gossip_heartbeat() (*libp2p.pubsub.gossipsub.GossipSub* method), 41
 gossip_history() (*libp2p.tools.constants.GossipsubParams* property), 67
 gossip_window() (*libp2p.tools.constants.GossipsubParams* property), 67
 GossipSub (class in *libp2p.pubsub.gossipsub*), 40
 GossipsubParams (class in *libp2p.tools.constants*), 66
 graft (*libp2p.pubsub.pb.rpc_pb2.ControlMessage* attribute), 35
 handle_dead_peer_queue() (*libp2p.pubsub.pubsub.Pubsub* method), 43
 handle_graft() (*libp2p.pubsub.gossipsub.GossipSub* method), 41
 handle_ihave() (*libp2p.pubsub.gossipsub.GossipSub* method), 41
 handle_incoming() (*libp2p.stream_muxer.mplex.mplex.Mplex* method), 61
 handle_iwant() (*libp2p.pubsub.gossipsub.GossipSub* method), 41
 handle_peer_queue() (*libp2p.pubsub.pubsub.Pubsub* method), 43
 handle_ping() (in module *libp2p.host.ping*), 11
 handle_prune() (*libp2p.pubsub.gossipsub.GossipSub* method), 41
 handle_rpc() (*libp2p.pubsub.abc.IPubsubRouter* method), 39
 handle_rpc() (*libp2p.pubsub.floodsub.FloodSub* method), 39
 handle_rpc() (*libp2p.pubsub.gossipsub.GossipSub* method), 41
 handle_subscription() (*libp2p.pubsub.pubsub.Pubsub* method), 44
 handlers (*libp2p.protocol_muxer.multiselect.Multiselect* attribute), 31
 handlers (*libp2p.protocol_muxer.multiselect_muxer_interface.IMultiselect* attribute), 34
 handshake() (*libp2p.protocol_muxer.multiselect.Multiselect* method), 31
 handshake() (*libp2p.protocol_muxer.multiselect_client.MultiselectClient* method), 32
 handshake() (*libp2p.protocol_muxer.multiselect_client_interface.IMultiselectClient* method), 33
 handshake_inbound() (*libp2p.security.noise.patterns.IPattern* method), 53
 handshake_inbound() (*libp2p.security.noise.patterns.PatternXX* method), 53
 handshake_outbound() (*libp2p.security.noise.patterns.IPattern* method), 53
 handshake_outbound() (*libp2p.security.noise.patterns.PatternXX* method), 53
 HandshakeFailure, 59
 HandshakeHasNotFinished, 51
 hash_type (*libp2p.crypto.authenticated_encryption.EncryptionParameters* attribute), 4
 hash_type (*libp2p.security.secio.transport.EncryptionParameters* attribute), 56
 hashes (*libp2p.security.secio.pb.spipe_pb2.Propose* attribute), 55
 hashes (*libp2p.security.secio.transport.Proposal* attribute), 56
 HeaderTags (class in *libp2p.stream_muxer.mplex.constants*), 60
 heartbeat() (*libp2p.pubsub.gossipsub.GossipSub* method), 41
 heartbeat_initial_delay

(*libp2p.pubsub.gossipsub.GossipSub* attribute), 41

heartbeat_initial_delay() (*libp2p.tools.constants.GossipsubParams* property), 67

heartbeat_interval (*libp2p.pubsub.gossipsub.GossipSub* attribute), 41

heartbeat_interval() (*libp2p.tools.constants.GossipsubParams* property), 67

history (*libp2p.pubsub.mcache.MessageCache* attribute), 43

history_size (*libp2p.pubsub.mcache.MessageCache* attribute), 43

host (*libp2p.pubsub.pubsub.Pubsub* attribute), 44

HostException, 10

I

IAddrBook (class in *libp2p.peer.addrbook_interface*), 22

IContentRouting (class in *libp2p.routing.interfaces*), 47

ID (class in *libp2p.peer.id*), 23

id (*libp2p.security.insecure.pb.plaintext_pb2.Exchange* attribute), 48

id_pubkey (*libp2p.security.noise.messages.NoiseHandshakePayload* attribute), 52

id_sig (*libp2p.security.noise.messages.NoiseHandshakePayload* attribute), 52

Identify (class in *libp2p.identity.identify.pb.identify_pb2*), 12

identify_handler_for() (in module *libp2p.identity.identify.protocol*), 13

identity_key (*libp2p.security.noise.pb.noise_pb2.NoiseHandshakePayload* attribute), 50

identity_sig (*libp2p.security.noise.pb.noise_pb2.NoiseHandshakePayload* attribute), 50

IdentityHash (class in *libp2p.peer.id*), 23

ihave (*libp2p.pubsub.pb.rpc_pb2.ControlMessage* attribute), 36

IHost (class in *libp2p.host.host_interface*), 10

IListener (class in *libp2p.transport.listener_interface*), 68

IMultiselectClient (class in *libp2p.protocol_muxer.multiselect_client_interface*), 33

IMultiselectCommunicator (class in *libp2p.protocol_muxer.multiselect_communicator_interface*), 34

IMultiselectMuxer (class in *libp2p.protocol_muxer.multiselect_muxer_interface*), 34

IMixedConn (class in *libp2p.stream_muxer.abc*), 64

IMixedStream (class in *libp2p.stream_muxer.abc*), 64

incoming_data_channel (*libp2p.stream_muxer.mplex.mplex_stream.MplexStream* attribute), 63

IncompatibleChoices, 55

IncompleteReadError, 14

InconsistentNonce, 55

INetConn (class in *libp2p.network.connection.net_connection_interface*), 16

INetStream (class in *libp2p.network.stream.net_stream_interface*), 18

INetwork (class in *libp2p.network.network_interface*), 18

INetworkService (class in *libp2p.network.network_interface*), 19

infer_local_type() (in module *libp2p.crypto.ecc*), 5

info_from_p2p_addr() (in module *libp2p.peer.peerinfo*), 26

initialize_pair() (in module *libp2p.crypto.authenticated_encryption*), 4

initiator_peers_queue (*libp2p.pubsub.pubsub_notifee.PubsubNotifee* attribute), 46

InvalidAddrError (class in *libp2p.network.notifee_interface*), 19

InvalidPayload (class in *libp2p.security.insecure.transport*), 49

InsecureTransport (class in *libp2p.security.insecure.transport*), 49

InvalidAddrError, 26

InvalidMACEException, 4

InvalidSignature, 51

InvalidSignatureOnExchange, 55

IPattern (class in *libp2p.security.noise.patterns*), 53

IPeerData (class in *libp2p.peer.peerdata_interface*), 25

IPeerMetadata (class in *libp2p.peer.peermetadata_interface*), 26

IPeerRouting (class in *libp2p.routing.interfaces*), 48

IPeerStore (class in *libp2p.peer.peerstore_interface*), 29

IPubsub (class in *libp2p.pubsub.abc*), 38

IPubsubRouter (class in *libp2p.pubsub.abc*), 38

IRawConnection (class in *libp2p.network.connection.raw_connection_interface*), 16

is_async() (*libp2p.pubsub.pubsub.TopicValidator* property), 45

is_closed() (*libp2p.network.connection.swarm_connection.SwarmConn* property), 16

is_closed() (*libp2p.stream_muxer.abc.IMuxedConn* property), 64
is_closed() (*libp2p.stream_muxer.mplex.mplex.Mplex* property), 62
is_initiator() (*libp2p.network.connection.raw_connection* attribute), 16
is_initiator() (*libp2p.network.connection.raw_connection_interface* attribute), 16
is_initiator() (*libp2p.security.secure_conn_interface.ISecureConn* attribute), 59
is_initiator() (*libp2p.stream_muxer.abc.IMuxedConn* property), 64
is_initiator() (*libp2p.stream_muxer.mplex.datastructures.StreamID* property), 61
is_initiator() (*libp2p.stream_muxer.mplex.mplex.Mplex* property), 62
is_initiator() (*libp2p.stream_muxer.mplex.mplex_stream_muxer* property), 63
is_valid_handshake() (in module *libp2p.protocol_muxer.multiselect*), 32
is_valid_handshake() (in module *libp2p.protocol_muxer.multiselect_client*), 32
ISecureConn (class in *libp2p.security.secure_conn_interface*), 59
ISecureTransport (class in *libp2p.security.secure_transport_interface*), 59
ISubscriptionAPI (class in *libp2p.pubsub.abc*), 39
ITransport (class in *libp2p.transport.transport_interface*), 69
iv (*libp2p.crypto.authenticated_encryption.EncryptionParameters* attribute), 4
iwant (*libp2p.pubsub.pb.rpc_pb2.ControlMessage* attribute), 36

J

join() (*libp2p.pubsub.abc.IPubsubRouter* method), 38
join() (*libp2p.pubsub.floodsub.FloodSub* method), 39
join() (*libp2p.pubsub.gossipsub.GossipSub* method), 41

K

Key (class in *libp2p.crypto.keys*), 6
key (*libp2p.pubsub.pb.rpc_pb2.Message* attribute), 36
KEY (*libp2p.pubsub.pb.rpc_pb2.TopicDescriptor.AuthOpts* attribute), 37
key_type (*libp2p.crypto.pb.crypto_pb2.PrivateKey* attribute), 3
key_type (*libp2p.crypto.pb.crypto_pb2.PublicKey* attribute), 3
keyHashes (*libp2p.pubsub.pb.rpc_pb2.TopicDescriptor.EncOpts* attribute), 37
KeyPair (class in *libp2p.crypto.keys*), 6

keys (*libp2p.pubsub.pb.rpc_pb2.TopicDescriptor.AuthOpts* attribute), 37
KeyType (class in *libp2p.crypto.keys*), 6
libp2p (module), 70
libp2p.crypto (module), 9
libp2p.crypto.authenticated_encryption (module), 4
libp2p.crypto.ecc (module), 4
libp2p.crypto.ed25519 (module), 5
libp2p.crypto.exceptions (module), 6
libp2p.crypto.key_exchange (module), 6
libp2p.crypto.keys (module), 6
libp2p.crypto.pb (module), 4
libp2p.crypto.pb.crypto_pb2 (module), 3
libp2p.crypto.rsa (module), 7
libp2p.crypto.secp256k1 (module), 8
libp2p.crypto.serialization (module), 8
libp2p.exceptions (module), 70
libp2p.host (module), 12
libp2p.host.basic_host (module), 9
libp2p.host.defaults (module), 10
libp2p.host.exceptions (module), 10
libp2p.host.host_interface (module), 10
libp2p.host.ping (module), 11
libp2p.host.routed_host (module), 12
libp2p.identity (module), 13
libp2p.identity.identify (module), 13
libp2p.identity.identify.pb (module), 13
libp2p.identity.identify.pb.identify_pb2 (module), 12
libp2p.identity.identify.protocol (module), 13
libp2p.io (module), 15
libp2p.io.abc (module), 13
libp2p.io.exceptions (module), 14
libp2p.io.msgio (module), 14
libp2p.io.utils (module), 15
libp2p.network (module), 22
libp2p.network.connection (module), 17
libp2p.network.connection.exceptions (module), 15
libp2p.network.connection.net_connection_interface (module), 16
libp2p.network.connection.raw_connection (module), 16

libp2p.network.connection.raw_connection_interface (module), 16

libp2p.network.connection.swarm_connection_interface (module), 16

libp2p.network.exceptions (module), 18

libp2p.network.network_interface (module), 18

libp2p.network.notifee_interface (module), 19

libp2p.network.stream (module), 18

libp2p.network.stream.exceptions (module), 17

libp2p.network.stream.net_stream (module), 17

libp2p.network.stream.net_stream_interface (module), 18

libp2p.network.swarm (module), 20

libp2p.peer (module), 31

libp2p.peer.addrbook_interface (module), 22

libp2p.peer.id (module), 23

libp2p.peer.peerdata (module), 23

libp2p.peer.peerdata_interface (module), 25

libp2p.peer.peerinfo (module), 26

libp2p.peer.peermetadata_interface (module), 26

libp2p.peer.peerstore (module), 26

libp2p.peer.peerstore_interface (module), 29

libp2p.protocol_muxer (module), 35

libp2p.protocol_muxer.exceptions (module), 31

libp2p.protocol_muxer.multiselect (module), 31

libp2p.protocol_muxer.multiselect_client_interface (module), 32

libp2p.protocol_muxer.multiselect_client_interface (module), 33

libp2p.protocol_muxer.multiselect_communicator_interface (module), 34

libp2p.protocol_muxer.multiselect_communicator_interface (module), 34

libp2p.protocol_muxer.multiselect_muxer_interface (module), 34

libp2p.pubsub (module), 47

libp2p.pubsub.abc (module), 38

libp2p.pubsub.exceptions (module), 39

libp2p.pubsub.floodsub (module), 39

libp2p.pubsub.gossipsub (module), 40

libp2p.pubsub.mcache (module), 42

libp2p.pubsub.pb (module), 38

libp2p.pubsub.pb.rpc_pb2 (module), 35

libp2p.pubsub.pubsub (module), 43

libp2p.pubsub.pubsub_notifee (module), 46

libp2p.pubsub.subscription (module), 47

libp2p.pubsub.validators (module), 47

libp2p.routing (module), 48

libp2p.routing.interfaces (module), 47

libp2p.security (module), 60

libp2p.security.base_session (module), 58

libp2p.security.base_transport (module), 58

libp2p.security.exceptions (module), 59

libp2p.security.insecure (module), 50

libp2p.security.insecure.pb (module), 49

libp2p.security.insecure.pb.plaintext_pb2 (module), 48

libp2p.security.insecure.transport (module), 49

libp2p.security.noise (module), 54

libp2p.security.noise.exceptions (module), 51

libp2p.security.noise.io (module), 51

libp2p.security.noise.messages (module), 52

libp2p.security.noise.patterns (module), 53

libp2p.security.noise.pb (module), 51

libp2p.security.noise.pb.noise_pb2 (module), 50

libp2p.security.noise.transport (module), 54

libp2p.security.secio (module), 58

libp2p.security.secio.exceptions (module), 55

libp2p.security.secio.pb (module), 55

libp2p.security.secio.pb.spipe_pb2 (module), 54

libp2p.security.secio.transport (module), 56

libp2p.security.secure_conn_interface (module), 59

libp2p.security.secure_transport_interface (module), 59

libp2p.security.secure_transport_interface (module), 59

libp2p.security_multistream (module), 59

libp2p.stream_muxer (module), 66

libp2p.stream_muxer.abc (module), 64

libp2p.stream_muxer.exceptions (module), 65

libp2p.stream_muxer.mplex (module), 64

libp2p.stream_muxer.mplex.constants (module), 60

libp2p.stream_muxer.mplex.datastructures (module), 61

libp2p.stream_muxer.mplex.exceptions (module), 61

- libp2p.stream_muxer.mplex.mplex (module), 61
- libp2p.stream_muxer.mplex.mplex_stream (module), 63
- libp2p.stream_muxer.muxer_multistream (module), 65
- libp2p.tools (module), 67
- libp2p.tools.constants (module), 66
- libp2p.tools.pubsub (module), 66
- libp2p.tools.pubsub.utils (module), 66
- libp2p.tools.utils (module), 67
- libp2p.transport (module), 70
- libp2p.transport.exceptions (module), 68
- libp2p.transport.listener_interface (module), 68
- libp2p.transport.tcp (module), 68
- libp2p.transport.tcp.tcp (module), 67
- libp2p.transport.transport_interface (module), 69
- libp2p.transport.typing (module), 69
- libp2p.transport.upgrader (module), 69
- libp2p.typing (module), 70
- libp2p.utils (module), 70
- libp2p_privkey (libp2p.security.noise.patterns.BasePattern attribute), 53
- libp2p_privkey (libp2p.security.noise.patterns.PatternXX attribute), 53
- libp2p_privkey (libp2p.security.noise.transport.Transport attribute), 54
- listen() (libp2p.network.network_interface.INetwork method), 18
- listen() (libp2p.network.notifee_interface.INotifee method), 20
- listen() (libp2p.network.swarm.Swarm method), 21
- listen() (libp2p.pubsub.pubsub_notifee.PubsubNotifee method), 46
- listen() (libp2p.transport.listener_interface.IListener method), 68
- listen() (libp2p.transport.tcp.tcp.TCPListener method), 68
- listen_addrs (libp2p.identity.identify.pb.identify_pb2.Identify attribute), 12
- listen_close() (libp2p.network.notifee_interface.INotifee method), 20
- listen_close() (libp2p.pubsub.pubsub_notifee.PubsubNotifee method), 46
- listener_nursery (libp2p.network.swarm.Swarm attribute), 21
- listeners (libp2p.network.network_interface.INetwork attribute), 19
- listeners (libp2p.network.network_interface.INetworkService attribute), 19
- listeners (libp2p.network.swarm.Swarm attribute), 21
- listeners (libp2p.transport.tcp.tcp.TCPListener attribute), 68
- local_encryption_parameters (libp2p.security.secio.transport.SessionParameters attribute), 57
- local_peer (libp2p.security.base_session.BaseSession attribute), 58
- local_peer (libp2p.security.insecure.transport.InsecureSession attribute), 49
- local_peer (libp2p.security.noise.patterns.BasePattern attribute), 53
- local_peer (libp2p.security.noise.patterns.PatternXX attribute), 53
- local_peer (libp2p.security.noise.transport.Transport attribute), 54
- local_peer (libp2p.security.secio.transport.SessionParameters attribute), 57
- local_private_key (libp2p.security.base_session.BaseSession attribute), 58
- local_private_key (libp2p.security.insecure.transport.InsecureSession attribute), 49
- mac_key (libp2p.crypto.authenticated_encryption.EncryptionParameters attribute), 4
- MacAndCipher (class in libp2p.crypto.authenticated_encryption), 4
- main() (in module examples.chat.chat), 73
- make_data_to_be_signed() (in module libp2p.security.noise.messages), 52
- make_exchange_message() (in module libp2p.security.insecure.transport), 50
- make_handshake_payload() (libp2p.security.noise.patterns.BasePattern method), 53
- make_handshake_payload_sig() (in module libp2p.security.noise.messages), 52
- make_pubsub_msg() (in module libp2p.tools.pubsub.utils), 66
- max_msg_size (libp2p.io.msgio.VarIntLengthMsgReadWrite attribute), 15
- max_msg_size (libp2p.security.insecure.transport.PlaintextHandshake attribute), 49
- mcache (libp2p.pubsub.gossipsub.GossipSub attribute), 42
- mesh (libp2p.pubsub.gossipsub.GossipSub attribute), 42
- mesh_heartbeat() (libp2p.pubsub.gossipsub.GossipSub method), 42
- Message (class in libp2p.pubsub.pb.rpc_pb2), 36
- message_all_peers() (libp2p.pubsub.pubsub.Pubsub method), 21

44

MessageCache (class in libp2p.pubsub.mcache), 42

messageIDs (libp2p.pubsub.pb.rpc_pb2.ControlIHave attribute), 35

messageIDs (libp2p.pubsub.pb.rpc_pb2.ControlIWant attribute), 35

MessageInitiator (libp2p.stream_muxer.mplex.constants attribute), 61

MessageReceiver (libp2p.stream_muxer.mplex.constants attribute), 61

MessageTooLarge, 14

metadata (libp2p.peer.peerdata.PeerData attribute), 24

mid (libp2p.pubsub.mcache.CacheEntry attribute), 42

MissingDeserializerError, 6

MissingLengthException, 14

MissingMessageException, 14

mode (libp2p.pubsub.pb.rpc_pb2.TopicDescriptor.AuthOpts attribute), 37

mode (libp2p.pubsub.pb.rpc_pb2.TopicDescriptor.EncOpts attribute), 37

Mplex (class in libp2p.stream_muxer.mplex.mplex), 61

MplexError, 61

MplexStream (class in libp2p.stream_muxer.mplex.mplex_stream), 63

MplexStreamClosed, 61

MplexStreamEOF, 61

MplexStreamReset, 61

MplexUnavailable, 61

MsgioException, 14

MsgReader (class in libp2p.io.abc), 13

MsgReadWriteCloser (class in libp2p.io.abc), 13

msgs (libp2p.pubsub.mcache.MessageCache attribute), 43

MsgWriter (class in libp2p.io.abc), 13

MultiError, 70

Multiselect (class in libp2p.protocol_muxer.multiselect), 31

multiselect (libp2p.host.basic_host.BasicHost attribute), 9

multiselect (libp2p.security.security_multistream.SecurityMultistream attribute), 60

multiselect (libp2p.stream_muxer_muxer_multistream.MuxerMultistream attribute), 65

multiselect_client (libp2p.host.basic_host.BasicHost attribute), 9

multiselect_client (libp2p.security.security_multistream.SecurityMultistream attribute), 60

multiselect_client (libp2p.stream_muxer_muxer_multistream.MuxerMultistream attribute), 65

MultiselectClient (class in libp2p.protocol_muxer_multiselect_client), 32

MultiselectClientError, 31

MultiselectCommunicator (class in libp2p.protocol_muxer_multiselect_communicator), 34

MultiselectCommunicatorError, 31

MultiselectError, 31

ms.HeaderTags (libp2p.network.connection.net_connection_interface.INetConnection attribute), 16

muxed_conn (libp2p.network.connection.swarm_connection.SwarmConnection attribute), 16

muxed_conn (libp2p.network.stream.net_stream_interface.INetStream attribute), 18

muxed_conn (libp2p.stream_muxer.abc.IMuxedStream attribute), 64

muxed_conn (libp2p.stream_muxer.mplex.mplex_stream.MplexStream attribute), 63

muxed_stream (libp2p.network.stream.net_stream.NetStream attribute), 17

MuxedConnError, 65

MuxedConnUnavailable, 65

MuxedStreamClosed, 65

MuxedStreamEOF, 65

MuxedStreamError, 65

MuxedStreamReset, 65

muxer_multistream (libp2p.transport.upgrader.TransportUpgrader attribute), 69

MuxerMultistream (class in libp2p.stream_muxer_muxer_multistream), 65

MuxerUpgradeFailure, 68

my_id() (libp2p.pubsub.abc.IPubsub property), 38

my_id() (libp2p.pubsub.pubsub.Pubsub property), 44

N

name (libp2p.pubsub.pb.rpc_pb2.TopicDescriptor attribute), 37

name (libp2p.stream_muxer.mplex.mplex_stream.MplexStream attribute), 63

negotiate (libp2p.protocol_muxer_multiselect.Multiselect method), 31

negotiate (libp2p.protocol_muxer_multiselect_muxer_interface.IMultiselect method), 34

NetStream (class in libp2p.network.stream.net_stream), 17

new() (libp2p.crypto.ecc.ECCPrivateKey class method), 4

new() (libp2p.crypto.ed25519.Ed25519PrivateKey class method), 5

new() (libp2p.crypto.rsa.RSAPrivateKey class method), 7

new () (*libp2p.crypto.secp256k1.Secp256k1PrivateKey* class method), 8
 new_conn () (*libp2p.stream_muxer.muxer_multistream.MultistreamTransportReadWriter* (class in *libp2p.security.noise.io*), 52
 method), 65
 new_host () (in module *libp2p*), 70
 new_stream () (*libp2p.host.basic_host.BasicHost* attribute), 9
 new_stream () (*libp2p.host.host_interface.IHost* attribute), 11
 new_stream () (*libp2p.network.connection.net_connection_interface.INetConn* attribute), 16
 new_stream () (*libp2p.network.connection.swarm_connection.SwarmConn* (class in *libp2p.network.swarm* attribute), 21
 method), 16
 new_stream () (*libp2p.network.network_interface.INetwork* attribute), 19
 new_stream () (*libp2p.network.swarm.Swarm* attribute), 21
 new_stream_receive_channel (*libp2p.stream_muxer.mplex.mplex.Mplex* attribute), 62
 new_stream_send_channel (*libp2p.stream_muxer.mplex.mplex.Mplex* attribute), 62
 new_swarm () (in module *libp2p*), 71
 NewStream (*libp2p.stream_muxer.mplex.constants.HeaderTags* attribute), 61
 next_channel_id (*libp2p.stream_muxer.mplex.mplex.Mplex* attribute), 62
 next_msg_len () (*libp2p.io.msgio.BaseMsgReadWriter* attribute), 14
 next_msg_len () (*libp2p.io.msgio.FixedSizeLenMsgReadWriter* attribute), 15
 next_msg_len () (*libp2p.io.msgio.VarIntLengthMsgReadWriter* attribute), 15
 noise_privkey (*libp2p.security.noise.transport.Transport* attribute), 54
 noise_state (*libp2p.security.noise.io.BaseNoiseMsgReadWriter* attribute), 51
 noise_state (*libp2p.security.noise.io.NoiseHandshakeReadWriter* attribute), 51
 noise_state (*libp2p.security.noise.io.NoiseTransportReadWriter* attribute), 52
 noise_static_key (*libp2p.security.noise.patterns.BasePattern* attribute), 53
 noise_static_key (*libp2p.security.noise.patterns.PatternXX* attribute), 53
 NoiseFailure, 51
 NoiseHandshakePayload (class in *libp2p.security.noise.messages*), 52
 NoiseHandshakePayload (class in *libp2p.security.noise.pb.noise_pb2*), 50
 NoiseHandshakeReadWriter (class in *libp2p.security.noise.io*), 51
 NoisePacketReadWriter (class in *libp2p.security.noise.io*), 51
 NoiseStateError, 51
 nonce (*libp2p.security.secio.transport.Proposal* attribute), 56
 NONE (*libp2p.pubsub.pb.rpc_pb2.TopicDescriptor.AuthOpts* attribute), 37
 NONE (*libp2p.pubsub.pb.rpc_pb2.TopicDescriptor.EncOpts* attribute), 37
 NoPubsubAttached, 39
 notify_closed_stream () (*libp2p.network.swarm.Swarm* method), 21
 notify_connected () (*libp2p.network.swarm.Swarm* method), 21
 notify_disconnected () (*libp2p.network.swarm.Swarm* method), 21
 notify_listen () (*libp2p.network.swarm.Swarm* method), 21
 notify_listen_close () (*libp2p.network.swarm.Swarm* method), 21
 notify_opened_stream () (*libp2p.network.swarm.Swarm* method), 21
 notify_subscriptions () (*libp2p.pubsub.pubsub.Pubsub* method), 44
 observed_addr (*libp2p.identity.identify.pb.identify_pb2.Identify* attribute), 12
 open_all_connect () (in module *libp2p.tools.pubsub.utils*), 66
 open_stream () (*libp2p.stream_muxer.abc.IMuxedConn* method), 64
 open_stream () (*libp2p.stream_muxer.mplex.mplex.Mplex* method), 62
 OpenStreamError, 68
 opened_stream () (*libp2p.network.notifee_interface.INotifee* method), 20
 opened_stream () (*libp2p.pubsub.pubsub_notifee.PubsubNotifee* method), 47
 order (*libp2p.security.secio.transport.SessionParameters* attribute), 57
P
 pack_control_msgs () (*libp2p.pubsub.gossipsub.GossipSub* method), 42

ParseError, 70
 PatternXX (class in libp2p.security.noise.patterns), 53
 peer_data_map (libp2p.peer.peerstore.PeerStore attribute), 28
 peer_id (libp2p.peer.peerinfo.PeerInfo attribute), 26
 peer_id (libp2p.stream_muxer.abc.IMuxedConn attribute), 64
 peer_id (libp2p.stream_muxer.mplex.mplex.Mplex attribute), 62
 peer_ids () (libp2p.peer.peerstore.PeerStore method), 28
 peer_ids () (libp2p.peer.peerstore_interface.IPeerStore method), 30
 peer_info () (libp2p.peer.peerstore.PeerStore method), 28
 peer_info () (libp2p.peer.peerstore_interface.IPeerStore method), 30
 peer_protocol (libp2p.pubsub.gossipsub.GossipSub attribute), 42
 peer_receive_channel (libp2p.pubsub.pubsub.Pubsub attribute), 44
 peer_topics (libp2p.pubsub.pubsub.Pubsub attribute), 44
 PeerData (class in libp2p.peer.peerdata), 23
 PeerDataError, 24
 PeerIDMismatchPubkey, 51
 PeerInfo (class in libp2p.peer.peerinfo), 26
 PeerMismatchException, 55
 peers (libp2p.pubsub.pubsub.Pubsub attribute), 44
 peers_with_addrs () (libp2p.peer.addrbook_interface.IAddrBook method), 23
 peers_with_addrs () (libp2p.peer.peerstore.PeerStore method), 28
 peers_with_addrs () (libp2p.peer.peerstore_interface.IPeerStore method), 30
 PeerStore (class in libp2p.peer.peerstore), 26
 peerstore (libp2p.host.basic_host.BasicHost attribute), 10
 peerstore (libp2p.network.network_interface.INetworkService attribute), 19
 peerstore (libp2p.network.network_interface.INetworkService attribute), 19
 peerstore (libp2p.network.swarm.Swarm attribute), 21
 PeerStoreError, 28
 permanent_public_key (libp2p.security.secio.transport.EncryptionParameters attribute), 56
 PlaintextHandshakeReadWriter (class in libp2p.security.insecure.transport), 49
 prefix (libp2p.security.noise.io.BaseNoiseMsgReadWriter attribute), 51
 pretty () (libp2p.peer.id.ID method), 23
 private_key (libp2p.crypto.keys.KeyPair attribute), 6
 PrivateKey (class in libp2p.crypto.keys), 6
 PrivateKey (class in libp2p.crypto.pb.crypto_pb2), 3
 privkey (libp2p.peer.peerdata.PeerData attribute), 24
 privkey () (libp2p.peer.peerstore.PeerStore method), 28
 privkey () (libp2p.peer.peerstore_interface.IPeerStore method), 30
 Proposal (class in libp2p.security.secio.transport), 56
 Propose (class in libp2p.security.secio.pb.spiffe_pb2), 55
 protocol_id (libp2p.network.stream.net_stream.NetStream attribute), 17
 protocol_name (libp2p.security.noise.patterns.BasePattern attribute), 53
 protocol_name (libp2p.security.noise.patterns.PatternXX attribute), 53
 protocol_version (libp2p.identity.identify.pb.identify_pb2.Identify attribute), 12
 protocols (libp2p.identity.identify.pb.identify_pb2.Identify attribute), 12
 protocols (libp2p.peer.peerdata.PeerData attribute), 24
 protocols (libp2p.pubsub.floodsub.FloodSub attribute), 40
 protocols (libp2p.pubsub.gossipsub.GossipSub attribute), 42
 protocols () (libp2p.pubsub.abc.IPubsub property), 38
 protocols () (libp2p.pubsub.pubsub.Pubsub property), 44
 provide () (libp2p.routing.interfaces.IContentRouting method), 48
 prune (libp2p.pubsub.pb.rpc_pb2.ControlMessage attribute), 36
 pubkey (libp2p.peer.peerdata.PeerData attribute), 24
 pubkey (libp2p.security.insecure.pb.plaintext_pb2.Exchange attribute), 48
 pubkey () (libp2p.peer.peerstore.PeerStore method), 28
 pubkey () (libp2p.peer.peerstore_interface.IPeerStore method), 30
 public_key (libp2p.crypto.keys.KeyPair attribute), 6
 public_key (libp2p.identity.identify.pb.identify_pb2.Identify attribute), 12
 public_key (libp2p.security.secio.pb.spiffe_pb2.Propose attribute), 55
 public_key (libp2p.security.secio.transport.Proposal attribute), 56
 PublicKey (class in libp2p.crypto.keys), 7
 PublicKey (class in libp2p.crypto.pb.crypto_pb2), 3
 publish (libp2p.pubsub.pb.rpc_pb2.RPC attribute), 37

publish() (*libp2p.pubsub.abc.IPubsub method*), 38
publish() (*libp2p.pubsub.abc.IPubsubRouter method*), 39
publish() (*libp2p.pubsub.floodsub.FloodSub method*), 40
publish() (*libp2p.pubsub.gossipsub.GossipSub method*), 42
publish() (*libp2p.pubsub.pubsub.Pubsub method*), 44
Pubsub (*class in libp2p.pubsub.pubsub*), 43
pubsub (*libp2p.pubsub.floodsub.FloodSub attribute*), 40
pubsub (*libp2p.pubsub.gossipsub.GossipSub attribute*), 42
PubsubNotiffee (*class in libp2p.pubsub.pubsub_notiffee*), 46
PubsubRouterError, 39
push_msg() (*libp2p.pubsub.pubsub.Pubsub method*), 44
put() (*libp2p.peer.peermetadata_interface.IPeerMetadata method*), 26
put() (*libp2p.peer.peerstore.PeerStore method*), 28
put() (*libp2p.peer.peerstore_interface.IPeerStore method*), 30
put() (*libp2p.pubsub.mcache.MessageCache method*), 43
put_metadata() (*libp2p.peer.peerdata.PeerData method*), 24
put_metadata() (*libp2p.peer.peerdata_interface.IPeerData method*), 25

R

rand (*libp2p.security.secio.pb.spice_pb2.Propose attribute*), 55
RawConnection (*class in libp2p.network.connection.raw_connection*), 16
RawConnError, 15
read() (*libp2p.io.abc.Reader method*), 14
read() (*libp2p.network.connection.raw_connection.RawConnection method*), 16
read() (*libp2p.network.stream.net_stream.NetStream method*), 17
read() (*libp2p.protocol_muxer.multiselect_communicator.MultiselectCommunicator method*), 34
read() (*libp2p.protocol_muxer.multiselect_communicator_interface.MultiselectCommunicator method*), 34
read() (*libp2p.security.insecure.transport.InsecureSession method*), 49
read() (*libp2p.stream_muxer.mplex.mplex_stream.MplexStream method*), 63
read_data() (*in module examples.chat.chat*), 73
read_deadline (*libp2p.stream_muxer.mplex.mplex_stream.MplexStream attribute*), 63
read_delim() (*in module libp2p.utils*), 70
read_exactly() (*in module libp2p.io.utils*), 15
read_length() (*in module libp2p.io.msgio*), 15
read_message() (*libp2p.stream_muxer.mplex.mplex.Mplex method*), 62
read_msg() (*libp2p.io.abc.MsgReader method*), 13
read_msg() (*libp2p.io.msgio.BaseMsgReadWriter method*), 14
read_msg() (*libp2p.security.noise.io.BaseNoiseMsgReadWriter method*), 51
read_msg() (*libp2p.security.secio.transport.SecioMsgReadWriter method*), 56
read_varint_prefixed_bytes() (*in module libp2p.utils*), 70
read_write_closer (*libp2p.io.msgio.BaseMsgReadWriter attribute*), 14
read_writer (*libp2p.protocol_muxer.multiselect_communicator.MultiselectCommunicator attribute*), 34
read_writer (*libp2p.security.noise.io.BaseNoiseMsgReadWriter attribute*), 51
read_writer (*libp2p.security.noise.io.NoiseHandshakeReadWriter attribute*), 51
read_writer (*libp2p.security.noise.io.NoiseTransportReadWriter attribute*), 52
read_writer (*libp2p.security.secio.transport.SecioMsgReadWriter attribute*), 56
ReadCloser (*class in libp2p.io.abc*), 13
Reader (*class in libp2p.io.abc*), 14
ReadWriteCloser (*class in libp2p.io.abc*), 13
ReadWriter (*class in libp2p.io.abc*), 13
receive_channel (*libp2p.pubsub.subscription.TrioSubscriptionAPI attribute*), 47
register_notiffee() (*libp2p.network.network_interface.INetwork method*), 19
register_notiffee() (*libp2p.network.swarm.Swarm method*), 21
remote_encryption_parameters (*libp2p.security.secio.transport.SessionParameters attribute*), 57
remote_peer (*libp2p.security.base_session.BaseSession attribute*), 58
remote_peer (*libp2p.security.insecure.transport.InsecureSession attribute*), 49
remote_peer (*libp2p.security.secio.transport.SessionParameters attribute*), 57
remote_permanent_pubkey (*libp2p.security.base_session.BaseSession attribute*), 58
remote_permanent_pubkey (*libp2p.security.insecure.transport.InsecureSession attribute*), 49
remove_conn() (*libp2p.network.swarm.Swarm method*), 22

remove_peer() (*libp2p.pubsub.abc.IPubsubRouter method*), 39
 remove_peer() (*libp2p.pubsub.floodsub.FloodSub method*), 40
 remove_peer() (*libp2p.pubsub.gossipsub.GossipSub method*), 42
 remove_stream() (*libp2p.network.connection.swarm.connection.SwarmConn method*), 17
 remove_topic_validator() (*libp2p.pubsub.abc.IPubsub method*), 38
 remove_topic_validator() (*libp2p.pubsub.pubsub.Pubsub method*), 44
 reset() (*libp2p.network.stream.net_stream.NetStream method*), 17
 reset() (*libp2p.network.stream.net_stream_interface.INetStream method*), 18
 reset() (*libp2p.stream_muxer.abc.IMuxedStream method*), 64
 reset() (*libp2p.stream_muxer.mplex.mplex_stream.MplexStream method*), 63
 ResetInitiator (*libp2p.stream_muxer.mplex.constants.HeaderTags attribute*), 61
 ResetReceiver (*libp2p.stream_muxer.mplex.constants.HeaderTags attribute*), 61
 RoutedHost (*class in libp2p.host.routed_host*), 12
 router (*libp2p.pubsub.pubsub.Pubsub attribute*), 44
 RPC (*class in libp2p.pubsub.pb.rpc_pb2*), 36
 RPC.SubOpts (*class in libp2p.pubsub.pb.rpc_pb2*), 36
 RSA (*libp2p.crypto.keys.KeyType attribute*), 6
 RSAPrivateKey (*class in libp2p.crypto.rsa*), 7
 RSAPublicKey (*class in libp2p.crypto.rsa*), 7
 run() (*in module examples.chat.chat*), 73
 run() (*libp2p.host.basic_host.BasicHost method*), 10
 run() (*libp2p.host.host_interface.IHost method*), 11
 run() (*libp2p.network.swarm.Swarm method*), 22
 run() (*libp2p.pubsub.gossipsub.GossipSub method*), 42
 run() (*libp2p.pubsub.pubsub.Pubsub method*), 44
 run_handshake() (*in module libp2p.security.insecure.transport*), 50

S

SecioException, 55
 SecioMsgReadWriter (*class in libp2p.security.secio.transport*), 56
 SecioPacketReadWriter (*class in libp2p.security.secio.transport*), 56
 Secp256k1 (*libp2p.crypto.keys.KeyType attribute*), 6
 Secp256k1PrivateKey (*class in libp2p.crypto.secp256k1*), 8
 Secp256k1PublicKey (*class in libp2p.crypto.secp256k1*), 8
 secure_inbound() (*libp2p.security.insecure.transport.InsecureTransport method*), 49
 secure_inbound() (*libp2p.security.noise.transport.Transport method*), 54
 secure_inbound() (*libp2p.security.secio.transport.Transport method*), 57
 secure_inbound() (*libp2p.security.secure_transport_interface.ISecureTransport method*), 59
 secure_inbound() (*libp2p.security.security_multistream.SecurityMultistream method*), 60
 secure_outbound() (*libp2p.security.insecure.transport.InsecureTransport method*), 49
 secure_outbound() (*libp2p.security.noise.transport.Transport method*), 54
 secure_outbound() (*libp2p.security.secio.transport.Transport method*), 57
 secure_outbound() (*libp2p.security.secure_transport_interface.ISecureTransport method*), 59
 secure_outbound() (*libp2p.security.security_multistream.SecurityMultistream method*), 60
 HeaderTags (*libp2p.stream_muxer.mplex.constants.HeaderTags attribute*), 62
 security_multistream (*libp2p.transport.upgrader.TransportUpgrader attribute*), 69
 SecurityMultistream (*class in libp2p.security.security_multistream*), 59
 SecurityUpgradeFailure, 68
 SedesException, 55
 seen_messages (*libp2p.pubsub.pubsub.Pubsub attribute*), 45
 select_from_minus() (*libp2p.pubsub.gossipsub.GossipSub static method*), 42
 select_one_of() (*libp2p.protocol_muxer.multiselect_client.MultiselectClient method*), 32
 select_one_of() (*libp2p.protocol_muxer.multiselect_client_interface.MultiselectClientInterface method*), 33
 select_transport() (*libp2p.security.security_multistream.SecurityMultistream method*), 60
 select_transport() (*libp2p.stream_muxer_muxer_multistream.MuxerMultistream method*), 65
 self_id (*libp2p.network.swarm.Swarm attribute*), 22
 SelfEncryption, 55
 send_message() (*libp2p.stream_muxer.mplex.mplex.Mplex method*), 62
 seqno (*libp2p.pubsub.pb.rpc_pb2.Message attribute*), 36
 serialize() (*libp2p.crypto.keys.PrivateKey method*), 36

7
 serialize() (*libp2p.crypto.keys.PublicKey* method), 7
 serialize() (*libp2p.security.noise.messages.NoiseHandshakePayload* method), 52
 serialize() (*libp2p.security.secio.transport.Proposal* method), 56
 SessionParameters (class in *libp2p.security.secio.transport*), 56
 set_deadline() (*libp2p.stream_muxer.abc.IMuxedStream* method), 64
 set_deadline() (*libp2p.stream_muxer.mplex.mplex_stream.MplexStream* method), 63
 set_protocol() (*libp2p.network.stream.net_stream.NetStream* method), 17
 set_protocol() (*libp2p.network.stream.net_stream_interface.INetStream* method), 18
 set_protocols() (*libp2p.peer.peerdata.PeerData* method), 24
 set_protocols() (*libp2p.peer.peerdata_interface.IPeerData* method), 25
 set_protocols() (*libp2p.peer.peerstore.PeerStore* method), 28
 set_protocols() (*libp2p.peer.peerstore_interface.IPeerStore* method), 30
 set_read_deadline() (*libp2p.stream_muxer.mplex.mplex_stream.MplexStream* method), 63
 set_stream_handler() (*libp2p.host.basic_host.BasicHost* method), 10
 set_stream_handler() (*libp2p.host.host_interface.IHost* method), 11
 set_stream_handler() (*libp2p.network.network_interface.INetwork* method), 19
 set_stream_handler() (*libp2p.network.swarm.Swarm* method), 22
 set_topic_validator() (*libp2p.pubsub.abc.IPubsub* method), 38
 set_topic_validator() (*libp2p.pubsub.pubsub.Pubsub* method), 45
 set_write_deadline() (*libp2p.stream_muxer.mplex.mplex_stream.MplexStream* method), 63
 sha256_digest() (in module *libp2p.peer.id*), 23
 shared_key (*libp2p.security.secio.transport.SessionParameters* attribute), 57
 SHAREDKEY (*libp2p.pubsub.pb.rpc_pb2.TopicDescriptor.Endpoint* attribute), 37
 shift() (*libp2p.pubsub.mcache.MessageCache* method), 43
 sign() (*libp2p.crypto.ecc.ECCPrivateKey* method), 4
 sign() (*libp2p.crypto.ed25519.Ed25519PrivateKey* method), 5
 sign() (*libp2p.crypto.keys.PrivateKey* method), 7
 sign() (*libp2p.crypto.rsa.RSAPrivateKey* method), 7
 sign() (*libp2p.crypto.secp256k1.Secp256k1PrivateKey* method), 8
 sign_key (*libp2p.pubsub.pubsub.Pubsub* attribute), 45
 signature (*libp2p.pubsub.pb.rpc_pb2.Message* attribute), 36
 signature (*libp2p.security.secio.pb.spice_pb2.Exchange* attribute), 54
 signature_validator() (in module *libp2p.pubsub.validators*), 47
 size_len_bytes (*libp2p.io.msgio.BaseMsgReadWriter* attribute), 14
 size_len_bytes (*libp2p.io.msgio.FixedSizeLenMsgReadWriter* attribute), 15
 size_len_bytes (*libp2p.security.noise.io.NoisePacketReadWriter* attribute), 52
 size_len_bytes (*libp2p.security.secio.transport.SecioPacketReadWriter* attribute), 56
 start() (*libp2p.network.connection.swarm_connection.SwarmConn* method), 17
 start() (*libp2p.stream_muxer.abc.IMuxedConn* method), 64
 start() (*libp2p.stream_muxer.mplex.mplex.MplexStream* method), 62
 stream (*libp2p.network.connection.raw_connection.RawConnection* attribute), 16
 stream_handler() (*libp2p.pubsub.pubsub.Pubsub* method), 45
 stream_id (*libp2p.stream_muxer.mplex.mplex_stream.MplexStream* attribute), 63
 StreamClosed, 17
 StreamEOF, 17
 StreamError, 17
 StreamFailure, 10
 StreamID (class in *libp2p.stream_muxer.mplex.datastructures*), 61
 StreamReset, 17
 streams (*libp2p.network.connection.swarm_connection.SwarmConn* attribute), 17
 streams (*libp2p.stream_muxer.mplex.mplex.Mplex* attribute), 62
 streams_lock (*libp2p.stream_muxer.mplex.mplex.MplexStream* attribute), 62
 streams_msg_channels (*libp2p.stream_muxer.mplex.mplex.Mplex* attribute), 62
 strict_signing (*libp2p.pubsub.pubsub.Pubsub* attribute), 45
 subscribe (*libp2p.pubsub.pb.rpc_pb2.RPC.SubOpts* attribute), 36

subscribe() (*libp2p.pubsub.abc.IPubsub* method), 38
 subscribe() (*libp2p.pubsub.pubsub.Pubsub* method), 45
 subscribed_topics_receive (*libp2p.pubsub.pubsub.Pubsub* attribute), 45
 subscribed_topics_send (*libp2p.pubsub.pubsub.Pubsub* attribute), 45
 subscriptions (*libp2p.pubsub.pb.rpc_pb2.RPC* attribute), 37
 Swarm (*class* in *libp2p.network.swarm*), 20
 swarm (*libp2p.network.connection.swarm_connection.SwarmConn* attribute), 17
 SwarmConn (*class* in *libp2p.network.connection.swarm_connection*), 16
 SwarmException, 18

T

TCP (*class* in *libp2p.transport.tcp.tcp*), 67
 TCPListener (*class* in *libp2p.transport.tcp.tcp*), 68
 time_to_live (*libp2p.pubsub.gossipsub.GossipSub* attribute), 42
 time_to_live() (*libp2p.tools.constants.GossipsubParameters* property), 67
 to_base58() (*libp2p.peer.id.ID* method), 23
 to_bytes() (*libp2p.crypto.ecc.ECCPrivateKey* method), 5
 to_bytes() (*libp2p.crypto.ecc.ECCPublicKey* method), 5
 to_bytes() (*libp2p.crypto.ed25519.Ed25519PrivateKey* method), 5
 to_bytes() (*libp2p.crypto.ed25519.Ed25519PublicKey* method), 5
 to_bytes() (*libp2p.crypto.keys.Key* method), 6
 to_bytes() (*libp2p.crypto.rsa.RSAPrivateKey* method), 7
 to_bytes() (*libp2p.crypto.rsa.RSAPublicKey* method), 7
 to_bytes() (*libp2p.crypto.secp256k1.Secp256k1PrivateKey* method), 8
 to_bytes() (*libp2p.crypto.secp256k1.Secp256k1PublicKey* method), 8
 to_bytes() (*libp2p.peer.id.ID* method), 23
 to_string() (*libp2p.peer.id.ID* method), 23
 topic_ids() (*libp2p.pubsub.abc.IPubsub* property), 38
 topic_ids() (*libp2p.pubsub.pubsub.Pubsub* property), 45
 topic_validators (*libp2p.pubsub.pubsub.Pubsub* attribute), 45

TopicDescriptor (*class* in *libp2p.pubsub.pb.rpc_pb2*), 37
 TopicDescriptor.AuthOpts (*class* in *libp2p.pubsub.pb.rpc_pb2*), 37
 TopicDescriptor.EncOpts (*class* in *libp2p.pubsub.pb.rpc_pb2*), 37
 topicID (*libp2p.pubsub.pb.rpc_pb2.ControlGraft* attribute), 35
 topicID (*libp2p.pubsub.pb.rpc_pb2.ControlIHave* attribute), 35
 topicID (*libp2p.pubsub.pb.rpc_pb2.ControlPrune* attribute), 36
 topicid (*libp2p.pubsub.pb.rpc_pb2.RPC.SubOpts* attribute), 36
 topicIDs (*libp2p.pubsub.pb.rpc_pb2.Message* attribute), 36
 topics (*libp2p.pubsub.mcache.CacheEntry* attribute), 42
 TopicValidator (*class* in *libp2p.pubsub.pubsub*), 45
 Transport (*class* in *libp2p.security.noise.transport*), 54
 Transport (*class* in *libp2p.security.secio.transport*), 57
 transport (*libp2p.network.swarm.Swarm* attribute), 22
 transports (*libp2p.security.security_multistream.SecurityMultistream* attribute), 60
 transports (*libp2p.stream_muxer_muxer_multistream.MuxerMultistream* attribute), 65
 TransportUpgrader (*class* in *libp2p.transport.upgrader*), 69
 TrioSubscriptionAPI (*class* in *libp2p.pubsub.subscription*), 47
 try_select() (*libp2p.protocol_muxer_multiselect_client.MultiselectClient* method), 32
 try_select() (*libp2p.protocol_muxer_multiselect_client_interface.IMultiselectClient* method), 33

U

unsubscribe() (*libp2p.pubsub.abc.IPubsub* method), 38
 unsubscribe() (*libp2p.pubsub.abc.ISubscriptionAPI* method), 39
 unsubscribe() (*libp2p.pubsub.pubsub.Pubsub* method), 45
 unsubscribe() (*libp2p.pubsub.subscription.TrioSubscriptionAPI* method), 47
 unsubscribe_fn (*libp2p.pubsub.subscription.TrioSubscriptionAPI* attribute), 47
 update() (*libp2p.peer.id.IdentityHash* method), 23
 upgrade_connection() (*libp2p.transport.upgrader.TransportUpgrader* method), 69

upgrade_listener() (libp2p.transport.upgrader.TransportUpgrader method), 69
 upgrade_security() (libp2p.transport.upgrader.TransportUpgrader method), 69
 UpgradeFailure, 68
 upgrader (libp2p.network.swarm.Swarm attribute), 22

V

validate_msg() (libp2p.pubsub.pubsub.Pubsub method), 45
 ValidationError, 70
 validator() (libp2p.pubsub.pubsub.TopicValidator property), 45
 VarIntLengthMsgReadWriter (class in libp2p.io.msgio), 15
 verify() (libp2p.crypto.ecc.ECCPublicKey method), 5
 verify() (libp2p.crypto.ed25519.Ed25519PublicKey method), 5
 verify() (libp2p.crypto.keys.PublicKey method), 7
 verify() (libp2p.crypto.rsa.RSAPublicKey method), 7
 verify() (libp2p.crypto.secp256k1.Secp256k1PublicKey method), 8
 verify_handshake_payload_sig() (in module libp2p.security.noise.messages), 52

W

wait_until_ready() (libp2p.pubsub.abc.IPubsub method), 38
 wait_until_ready() (libp2p.pubsub.pubsub.Pubsub method), 45
 window() (libp2p.pubsub.mcache.MessageCache method), 43
 window_size (libp2p.pubsub.mcache.MessageCache attribute), 43
 with_noise_pipes (libp2p.security.noise.transport.Transport attribute), 54
 WOT (libp2p.pubsub.pb.rpc_pb2.TopicDescriptor.AuthOpts attribute), 37
 WOT (libp2p.pubsub.pb.rpc_pb2.TopicDescriptor.EncOpts attribute), 37
 write() (libp2p.io.abc.Writer method), 14
 write() (libp2p.network.connection.raw_connection.RawConnection method), 16
 write() (libp2p.network.stream.net_stream.NetStream method), 17
 write() (libp2p.protocol_muxer.multiselect_communicator.MultiselectCommunicator method), 34
 write() (libp2p.protocol_muxer.multiselect_communicator_interface.IMultiselectCommunicator method), 34
 write() (libp2p.security.insecure.transport.InsecureSession method), 49
 write() (libp2p.stream_muxer.mplex.mplex_stream.MplexStream method), 63
 write_data() (in module examples.chat.chat), 73
 write_deadline (libp2p.stream_muxer.mplex.mplex_stream.MplexStream attribute), 63
 write_msg() (libp2p.io.abc.MsgWriter method), 13
 write_msg() (libp2p.io.msgio.BaseMsgReadWriter method), 15
 write_msg() (libp2p.security.noise.io.BaseNoiseMsgReadWriter method), 51
 write_msg() (libp2p.security.secio.transport.SecioMsgReadWriter method), 56
 write_to_stream() (libp2p.stream_muxer.mplex.mplex.Mplex method), 62
 WriteCloser (class in libp2p.io.abc), 14
 Writer (class in libp2p.io.abc), 14

X

xor_id() (libp2p.peer.id.ID property), 23