
i2plib Documentation

Release 0.0.14

Viktor Villainov

Feb 05, 2019

Contents

1	Contents	3
1.1	Quick start	3
1.2	Developer Interface	5
2	Resources	11
3	Indices and tables	13
	Python Module Index	15

i2plib is a modern asynchronous library for building I2P applications.

1.1 Quick start

1.1.1 Installing

```
pip install i2plib
```

Requirements:

- Python version ≥ 3.5
- I2P router with SAM API enabled

1.1.2 Connecting to a remote I2P destination

```
import asyncio
import i2plib

async def connect_test(destination):
    session_name = "test-connect"

    # create a SAM stream session
    await i2plib.create_session(session_name)

    # connect to a destination
    reader, writer = await i2plib.stream_connect(session_name, destination)

    # write data to a socket
    writer.write(b"PING")

    # asynchronously receive data
    data = await reader.read(4096)
```

(continues on next page)

(continued from previous page)

```
print(data.decode())

# close the connection
writer.close()

# run event loop
loop = asyncio.get_event_loop()
loop.run_until_complete(connect_test("dummy.i2p"))
loop.stop()
```

1.1.3 Accept connections in I2P

```
import asyncio
import i2plib

async def accept_test():
    session_name = "test-accept"

    # create a SAM stream session
    await i2plib.create_session(session_name)

    # accept a connection
    reader, writer = await i2plib.stream_accept(session_name)

    # first string on a client connection always contains clients I2P destination
    dest = await reader.readline()
    remote_destination = i2plib.Destination(dest.decode().strip())

    # read for the actual incoming data from the client
    data = await reader.read(4096)

    print(data.decode())

    # send data back
    writer.write(b"PONG")

    # close the connection
    writer.close()

# run event loop
loop = asyncio.get_event_loop()
loop.run_until_complete(accept_test())
loop.stop()
```

1.1.4 Server tunnel

Expose a local service to I2P like that:

```
import asyncio
import i2plib

loop = asyncio.get_event_loop()
# making your local web server available in the I2P network
```

(continues on next page)

(continued from previous page)

```
tunnel = i2plib.ServerTunnel(("127.0.0.1", 80))
asyncio.ensure_future(tunnel.run())

try:
    loop.run_forever()
except KeyboardInterrupt:
    pass
finally:
    loop.close()
```

1.1.5 Client tunnel

Bind a remote I2P destination to a port on your local host:

```
import asyncio
import i2plib

loop = asyncio.get_event_loop()
# bind irc.echelon.i2p to 127.0.0.1:6669
tunnel = i2plib.ClientTunnel("irc.echelon.i2p", ("127.0.0.1", 6669))
asyncio.ensure_future(tunnel.run())

try:
    loop.run_forever()
except KeyboardInterrupt:
    pass
finally:
    loop.close()
```

1.1.6 More examples

You can see more demo applications in *docs/examples* directory of the source repository.

1.2 Developer Interface

This part of the documentation covers all the interfaces of i2plib.

1.2.1 Network connections

These 4 *coroutines* provide everything you need for making connections inside I2P network. All of them return a tuple of transports (*reader*, *writer*) to deal with.

The *reader* returned is an `asyncio.StreamReader` instance; the *writer* is an `asyncio.StreamWriter` instance.

```
i2plib.create_session(session_name, sam_address=('127.0.0.1', 7656), loop=None,
                    style='STREAM', signature_type=7, destination=None, options={})
```

A coroutine used to create a new SAM session.

Parameters

- **session_name** – Session nick name

- **sam_address** – (optional) SAM API address
- **loop** – (optional) Event loop instance
- **style** – (optional) Session style, can be STREAM, DATAGRAM, RAW
- **signature_type** – (optional) If the destination is TRANSIENT, this signature type is used
- **destination** – (optional) Destination to use in this session. Can be a base64 encoded string, *i2plib.Destination* instance or None. TRANSIENT destination is used when it is None.
- **options** – (optional) A dict object with i2cp options

Returns A (reader, writer) pair

`i2plib.stream_connect(session_name, destination, sam_address=('127.0.0.1', 7656), loop=None)`
A coroutine used to connect to a remote I2P destination.

Parameters

- **session_name** – Session nick name
- **destination** – I2P destination to connect to
- **sam_address** – (optional) SAM API address
- **loop** – (optional) Event loop instance

Returns A (reader, writer) pair

`i2plib.stream_accept(session_name, sam_address=('127.0.0.1', 7656), loop=None)`
A coroutine used to accept a connection from the I2P network.

Parameters

- **session_name** – Session nick name
- **sam_address** – (optional) SAM API address
- **loop** – (optional) Event loop instance

Returns A (reader, writer) pair

`i2plib.get_sam_socket(sam_address=('127.0.0.1', 7656), loop=None)`
A coroutine used to create a new SAM socket.

Parameters

- **sam_address** – (optional) SAM API address
- **loop** – (optional) event loop instance

Returns A (reader, writer) pair

1.2.2 Context managers

The following are asynchronous context managers for making I2P connections.

You can use them like that:

```

import asyncio
import i2plib

async def connect_test(destination):
    session_name = "test"

    async with i2plib.Session(session_name):
        async with i2plib.StreamConnection(session_name, destination) as c:
            c.write(b"PING")
            resp = await c.read(4096)

    print(resp)

loop = asyncio.get_event_loop()
loop.run_until_complete(connect_test("dummy.i2p"))
loop.stop()

```

class `i2plib.Session`(*session_name*, *sam_address*=('127.0.0.1', 7656), *loop*=None, *style*=*'STREAM'*, *signature_type*=7, *destination*=None, *options*={})
 Async SAM session context manager.

Parameters

- **session_name** – Session nick name
- **sam_address** – (optional) SAM API address
- **loop** – (optional) Event loop instance
- **style** – (optional) Session style, can be STREAM, DATAGRAM, RAW
- **signature_type** – (optional) If the destination is TRANSIENT, this signature type is used
- **destination** – (optional) Destination to use in this session. Can be a base64 encoded string, `i2plib.Destination` instance or None. TRANSIENT destination is used when it is None.
- **options** – (optional) A dict object with i2cp options

Returns `i2plib.Session` object

class `i2plib.StreamConnection`(*session_name*, *destination*, *sam_address*=('127.0.0.1', 7656), *loop*=None)
 Async stream connection context manager.

Parameters

- **session_name** – Session nick name
- **destination** – I2P destination to connect to
- **sam_address** – (optional) SAM API address
- **loop** – (optional) Event loop instance

Returns `i2plib.StreamConnection` object

class `i2plib.StreamAcceptor`(*session_name*, *sam_address*=('127.0.0.1', 7656), *loop*=None)
 Async stream acceptor context manager.

Parameters

- **session_name** – Session nick name

- **sam_address** – (optional) SAM API address
- **loop** – (optional) Event loop instance

Returns *i2plib.StreamAcceptor* object

1.2.3 Utilities

`i2plib.dest_lookup(domain, sam_address=('127.0.0.1', 7656), loop=None)`

A coroutine used to lookup a full I2P destination by .i2p domain or .b32.i2p address.

Parameters

- **domain** – Address to be resolved, can be a .i2p domain or a .b32.i2p address.
- **sam_address** – (optional) SAM API address
- **loop** – (optional) Event loop instance

Returns An instance of *i2plib.Destination*

`i2plib.new_destination(sam_address=('127.0.0.1', 7656), loop=None, sig_type=7)`

A coroutine used to generate a new destination with a private key of a chosen signature type.

Parameters

- **sam_address** – (optional) SAM API address
- **loop** – (optional) Event loop instance
- **sig_type** – (optional) Signature type

Returns An instance of *i2plib.Destination*

`i2plib.get_sam_address()`

Get SAM address from environment variable I2P_SAM_ADDRESS, or use a default value

1.2.4 Tunnel API

Tunnel API is the quickest way to use regular software inside I2P. Client tunnel binds a remote I2P destination to a local address. Server tunnel exposes a local address to the I2P network.

class `i2plib.tunnel.I2PTunnel(local_address, destination=None, session_name=None, options={}, loop=None, sam_address=('127.0.0.1', 7656))`

Base I2P Tunnel object, not to be used directly

Parameters

- **local_address** – A local address to use for a tunnel. E.g. ("127.0.0.1", 6668)
- **destination** – (optional) Destination to use for this tunnel. Can be a base64 encoded string, *i2plib.Destination* instance or None. A new destination is created when it is None.
- **session_name** – (optional) Session nick name. A new session nickname is generated if not specified.
- **options** – (optional) A dict object with i2cp options
- **loop** – (optional) Event loop instance
- **sam_address** – (optional) SAM API address

stop()
Stop the tunnel

class i2plib.ClientTunnel (*remote_destination*, *args, **kwargs)
Client tunnel, a subclass of i2plib.tunnel.I2PTunnel

If you run a client tunnel with a local address (“127.0.0.1”, 6668) and a remote destination “irc.echelon.i2p”, all connections to 127.0.0.1:6668 will be proxied to irc.echelon.i2p.

Parameters **remote_destination** – Remote I2P destination, can be either .i2p domain, .b32.i2p address, base64 destination or *i2plib.Destination* instance

run()
A coroutine used to run the tunnel

stop()
Stop the tunnel

class i2plib.ServerTunnel (*args, **kwargs)
Server tunnel, a subclass of i2plib.tunnel.I2PTunnel

If you want to expose a local service 127.0.0.1:80 to the I2P network, run a server tunnel with a local address (“127.0.0.1”, 80). If you don’t provide a private key or a session name, it will use a TRANSIENT destination.

run()
A coroutine used to run the tunnel

stop()
Stop the tunnel

1.2.5 Data structures

class i2plib.Destination (*data=None, path=None, has_private_key=False*)
I2P destination

<https://geti2p.net/spec/common-structures#destination>

Parameters

- **data** – (optional) Base64 encoded data or binary data
- **path** – (optional) A path to a file with binary data
- **has_private_key** – (optional) Does data have a private key?

base32
Base32 destination hash of this destination

base64 = None
Base64 encoded destination

data = None
Binary destination

private_key = None
i2plib.PrivateKey instance or None

class i2plib.PrivateKey (*data*)
I2P private key

<https://geti2p.net/spec/common-structures#keysandcert>

Parameters **data** – Base64 encoded data or binary data

base64 = None
Base64 encoded private key

data = None
Binary private key

1.2.6 Exceptions

exception i2plib.CantReachPeer
The peer exists, but cannot be reached

exception i2plib.DuplicatedDest
The specified Destination is already in use

exception i2plib.DuplicatedId
The nickname is already associated with a session

exception i2plib.I2PError
A generic I2P error

exception i2plib.InvalidId
STREAM SESSION ID doesn't exist

exception i2plib.InvalidKey
The specified key is not valid (bad format, etc.)

exception i2plib.KeyNotFound
The naming system can't resolve the given name

exception i2plib.PeerNotFound
The peer cannot be found on the network

exception i2plib.Timeout
The peer cannot be found on the network

CHAPTER 2

Resources

- [i2plib online documentation](#)
- [Invisible Internet Project](#)
- [SAM API documentation](#)
- [Python asyncio documentation](#)

CHAPTER 3

Indices and tables

- `genindex`
- `modindex`
- `search`

i

i2plib, 5

B

base32 (i2plib.Destination attribute), 9
base64 (i2plib.Destination attribute), 9
base64 (i2plib.PrivateKey attribute), 9

C

CantReachPeer, 10
ClientTunnel (class in i2plib), 9
create_session() (in module i2plib), 5

D

data (i2plib.Destination attribute), 9
data (i2plib.PrivateKey attribute), 10
dest_lookup() (in module i2plib), 8
Destination (class in i2plib), 9
DuplicatedDest, 10
DuplicatedId, 10

G

get_sam_address() (in module i2plib), 8
get_sam_socket() (in module i2plib), 6

I

I2PError, 10
i2plib (module), 5
I2PTunnel (class in i2plib.tunnel), 8
InvalidId, 10
InvalidKey, 10

K

KeyNotFound, 10

N

new_destination() (in module i2plib), 8

P

PeerNotFound, 10
private_key (i2plib.Destination attribute), 9

PrivateKey (class in i2plib), 9

R

run() (i2plib.ClientTunnel method), 9
run() (i2plib.ServerTunnel method), 9

S

ServerTunnel (class in i2plib), 9
Session (class in i2plib), 7
stop() (i2plib.ClientTunnel method), 9
stop() (i2plib.ServerTunnel method), 9
stop() (i2plib.tunnel.I2PTunnel method), 8
stream_accept() (in module i2plib), 6
stream_connect() (in module i2plib), 6
StreamAcceptor (class in i2plib), 7
StreamConnection (class in i2plib), 7

T

Timeout, 10