leaflet Documentation

Release 0.0.4.dev1

MuxZeroNet

Dec 10, 2017

Contents

Leaflet is a Python library you can use to build applications that routes packets through the I2P anonymous network. Leaflet is a client of I2P's Simple Anonymous Messaging v3.0 protocol.

Leaflet is based on `i2p.socket` but has a different level of abstraction and provides more Pythonic API. Leaflet is written in Python 3 and licensed under the MIT/Expat license.

CHAPTER 1

Table of contents

1.1 Writing a server using stream sockets

Make a Controller object.

```
controller = Controller()
```

Create our Destination. Our Destination is our "IP Address" in the I2P network.

Our Destination instance will be discarded once it gets garbage collected, so we should keep the reference to it, and use it in a context manager.

```
with controller.create_dest() as our_dest:
    print('Server address: ' + our_dest.base32 + '.b32.i2p')
```

Tell SAM we need to accept connections. The method `our_dest.register_accept` tells SAM to accept connections. The return value `conn` is a socket object. This method will not block.

```
    conn = our_dest.register_accept()
```

SAM will write to the `conn` socket only when a data stream comes in. Wait until a message comes in. When a message came in, read and strip the SAM response headers.

The method `conn.parse_headers` will block until a message comes in.

Now a message comes in, but first we need to strip the SAM headers.

```
    addr = conn.parse_headers()
```

Now we have the connection and the address. Pass them to a handler.

```
    handler(addr, conn)
```

Write the handler.

```
def handler(addr, conn):
    # now we can read the real message
    request = conn.recv(4096)
    print('Received a message from %r: %r' % (addr, request))
    # reply
    conn.sendall(b'Hello, how are you?')
    conn.close()
```

Write our client:

```
def send(server_addr):
    # test SAM connection
    controller = Controller()

    with controller.create_dest() as our_dest:
        # connect to a remote destination and send our message
        sock = our_dest.connect(server_addr)
        # SAM will give us response headers
        # when the connection is successful
        sam_reply = sock.parse_headers()
        # now we can send data
        sock.sendall(b'Hello, there!')
        real_reply = sock.recv(4096)

        print(sam_reply, real_reply)
        sock.close()

send('serveraddress.b32.i2p')
```

1.2 Class reference

Leaflet defines the following public classes and functions.

1.2.1 Controller and our Destination

class **Controller**(*object*)

__init__(*self*, *sam_timeout=60.0*, *sam_api=('127.0.0.1', 7656)*, *dgram_api=('127.0.0.1', 7655)*, *max_version='3.0'*)
Make a SAM Controller instance using the given information, and test SAM connection.

check_api(*self*)
Check SAM API connection. This method will be automatically called when the constructor is called.

Raises **OSError** – if failed to connect to SAM API.

lookup(*self*, *name*)
Lookup and return the full Destination of the I2P domain name. If *name* is a *Dest* instance, return it directly.

Parameters **name** (*str or Dest*) – the .b32.i2p domain name.

Returns A *Dest* instance.

Raises ***NSError*** – if lookup failed.

create_dest (*self*, *name = None*, *style='stream'*, *forward = None*, *i2cp = None*)
Create an ephemeral Destination. The Destination will be destroyed when dereferenced.

Parameters

- **name** (*str or None*) – a human-readable "nickname" of our Destination, must be unique and cannot contain whitespaces. If the name is not provided, a random name will be generated.
- **style** (*str*) – the Destination type, can be either `stream` or `datagram`.
- **forward** (*int, tuple or None*) – for `datagram` Destinations only. The port number or endpoint which SAM will forward incoming datagram to.
- **i2cp** (*dict or None*) – additional I2CP options.

Returns An *OurDest* instance.

Raises

- ***CreateDestError*** – if failed to create an ephemeral Destination.
- ***HandshakeError*** – if handshake failed.

class **OurDest** (*Dest*)
Internal class returned when calling *Controller.create_dest()*. It inherits methods from the *Dest* class. It also defines the following methods.

connect (*self*, *other*)
For `stream` Destinations only.

Ask SAM to connect to a remote I2P peer, and write to the returned wrapped socket object when the connection is successful.

Because of how the SAM protocol is designed, this method simply sends a `STREAM CONNECT` request to the SAM port. This method will not block. To receive and parse the SAM reply headers, use *StreamSocket.parse_headers()* immediately.

Parameters **other** (*str or Dest*) – the remote I2P peer to connect to.

Returns a *StreamSocket* instance.

Raises ***HandshakeError*** – if handshake failed.

register_accept (*self*)
For `stream` Destinations only.

Ask SAM to accept connections to this destination, and write to the returned wrapped socket object when a data stream is available.

Because of how the SAM protocol is designed, this method simply sends a `STREAM ACCEPT` request to the SAM port. This method will not block. To receive and parse the SAM reply headers, use *StreamSocket.parse_headers()* immediately.

Returns a *StreamSocket* instance.

Raises

- ***HandshakeError*** – if handshake failed.
- ***AcceptError*** – if failed to accept connections.

bind (*self*)
For `datagram` Destinations only.

Make a datagram socket, and bind the datagram socket to the endpoint specified in the *forward* parameter, in the constructor, returning the datagram socket.

Returns a *DatagramSocket* instance.

Raises **OSError** – when failed to bind to the given endpoint.

close(*self*)
Close the SAM connection and free resources, destroying the ephemeral Destination.

__enter__(*self*)

__exit__(*self*, **args*, ***kwargs*)
Allows you to use *Controller.create_dest()* inside a `with` statement suite.

```
controller = Controller()
with controller.create_dest() as our_dest:
    do_stuff()
```

1.2.2 Wrapped socket

class WrappedSocket(*object*)

class StreamSocket(*WrappedSocket*)
A wrapped socket object that exposes I2P concepts instead of IP concepts. It defines the following methods.

parse_headers(*self*)
Receive and parse SAM reply headers. This method will block or raise BlockingIOError or raise socket.timeout depending on your socket settings.

When this method is used immediately after *OurDest.connect()*, the return value and exception type are the following.

Returns a *SAMReply* instance.

Raises ***ReachError*** – when the remote peer was unreachable.

When this method is used immediately after *OurDest.register_accept()*, the return value and exception type are the following.

Returns a *Dest* instance, indicating the source of the packet.

Raises **ValueError** – if the Destination in the SAM reply headers cannot be parsed.

lookup(*self*, *name*)
The alternative to the *gethostbyname* method.

See *Controller.lookup()*

close(*self*)

__enter__(*self*)

__exit__(*self*, **args*, ***kwargs*)

Note: It has the following passthru methods.

```
type proto send recv sendall sendfile fileno shutdown detach makefile
setsockopt getsockopt setblocking settimeout
```

Notably, the following methods are not available. Calling any of them results in an AttributeError. Instead, their alternatives should be used.

bind()

listen()

accept()

connect()

gethostbyname()

gethostbyname_ex()

class DatagramSocket(*WrappedSocket*)

A wrapped datagram socket object that exposes I2P concepts instead of IP concepts. It defines the following methods.

class transmit(*self*, *data*[, *flags*], *dest*)

The alternative to the *sendto* method.

Parameters

- **data** (*bytes*) – the payload, excluding the SAM datagram header.
- **dest** (*str or Dest*) – where to send the payload to.

Raises

- ***NSError*** – if naming lookup failed.
- **OSError** – if a socket error ocurred.

Note: The maximum acceptable payload size is ~31 KB. However, it is recommended to keep the payload size under 15 KB.

---

class collect(*self*, *bufsize=32*1024*, **args*)

The alternative to the *recvfrom* method.

Parameters **bufsize** (*int*) – the number of bytes you want to receive, excluding the SAM datagram header.

Returns a `(data, address)` -> (bytes, *Dest*) tuple.

Raises ***SourceError*** – if the packet is not forwarded by SAM.

Notably, the following methods are not available. Calling any of them results in an AttributeError. Instead, their alternatives should be used.

sendto()

recvfrom()

1.2.3 SAM data structure

class Dest(*object*)

A parsed Destination or KeyFile.

Variables

- **is_private** (*bool*) – *True* if the Destination contains private keys.
- **base64** (*str*) – Base-64 representation of the public component of the Destination.
- **base32** (*str*) – Base-32 representation of the SHA-256 hash of the public component of the Destination.

__init__(*self*, *keyfile*, *encoding*, *sig_type = None*, *private=False*)

class SAMReply(*object*)

1.2.4 Exceptions

class HandshakeError(*OSError*)

class NSError(*OSError*)

class CreateDestError(*OSError*)

class ReachError(*OSError*)

class AcceptError(*OSError*)

class SourceError(*OSError*)

1.3 Examples

Basic stream server and client: `leaflet/examples/basic.py`

```
python3 -m leaflet.examples.basic server
```

```
python3 -m leaflet.examples.basic client server.b32.i2p
```

Datagram server and client: `leaflet/examples/datagram.py`

```
python3 -m leaflet.examples.datagram server
```

```
python3 -m leaflet.examples.datagram client server.b32.i2p
```

Index