aioice Documentation

Jeremy Lainé

Apr 01, 2023

CONTENTS

`aioice` is a library for Interactive Connectivity Establishment (RFC 5245) in Python. It is built on top of `asyncio`, Python's standard asynchronous I/O framework.

Interactive Connectivity Establishment (ICE) is useful for applications that establish peer-to-peer UDP data streams, as it facilitates NAT traversal. Typical usecases include SIP and WebRTC.

CHAPTER

ONE

API REFERENCE

class aioice.**Connection**(*ice_controlling*, *components=1*, *stun_server=None*, *turn_server=None*, *turn_username=None*, *turn_password=None*, *turn_ssl=False*, *turn_transport='udp'*, *use_ipv4=True*, *use_ipv6=True*, *transport_policy=TransportPolicy.ALL*)

An ICE connection for a single media stream.

Parameters

- **ice_controlling** (bool) – Whether the local peer has the controlling role.
- **components** (int) – The number of components.
- **stun_server** (Optional[Tuple[str, int]]) – The address of the STUN server or *None*.
- **turn_server** (Optional[Tuple[str, int]]) – The address of the TURN server or *None*.
- **turn_username** (Optional[str]) – The username for the TURN server.
- **turn_password** (Optional[str]) – The password for the TURN server.
- **turn_ssl** (bool) – Whether to use TLS for the TURN server.
- **turn_transport** (str) – The transport for TURN server, *"udp"* or *"tcp"*.
- **use_ipv4** (bool) – Whether to use IPv4 candidates.
- **use_ipv6** (bool) – Whether to use IPv6 candidates.
- **transport_policy** (TransportPolicy) – Transport policy.

async add_remote_candidate(*remote_candidate*)

Add a remote candidate or signal end-of-candidates.

To signal end-of-candidates, pass *None*.

Parameters
remote_candidate (*Candidate*) – A *Candidate* instance or *None*.

Return type
None

async gather_candidates()

Gather local candidates.

You **must** call this coroutine before calling *connect()*.

Return type
None

get_default_candidate(*component*)

Get the default local candidate for the specified component.

Parameters
component (int) – The component whose default candidate is requested.

Return type
Optional[*Candidate*]

async connect()

Perform ICE handshake.

This coroutine returns if a candidate pair was successfuly nominated and raises an exception otherwise.

Return type
None

async close()

Close the connection.

Return type
None

async recv()

Receive the next datagram.

The return value is a *bytes* object representing the data received.

If the connection is not established, a *ConnectionError* is raised.

Return type
bytes

async recvfrom()

Receive the next datagram.

The return value is a *(bytes, component)* tuple where *bytes* is a bytes object representing the data received and *component* is the component on which the data was received.

If the connection is not established, a *ConnectionError* is raised.

Return type
Tuple[bytes, int]

async send(*data*)

Send a datagram on the first component.

If the connection is not established, a *ConnectionError* is raised.

Parameters
data (bytes) – The data to be sent.

Return type
None

async sendto(*data*, *component*)

Send a datagram on the specified component.

If the connection is not established, a *ConnectionError* is raised.

Parameters

- **data** (bytes) – The data to be sent.

- **component** (int) – The component on which to send the data.

Return type
None

set_selected_pair(*component*, *local_foundation*, *remote_foundation*)

Force the selected candidate pair.

If the remote party does not support ICE, you should using this instead of calling *connect()*.

Return type
None

property local_candidates: List[Candidate]

Local candidates, automatically set by *gather_candidates()*.

local_password

Local password, automatically set to a random value.

local_username

Local username, automatically set to a random value.

property remote_candidates: List[Candidate]

Remote candidates, which you need to populate using *add_remote_candidate()*.

remote_password: Optional[str]

Remote password, which you need to set.

remote_username: Optional[str]

Remote username, which you need to set.

class aioice.**Candidate**(*foundation*, *component*, *transport*, *priority*, *host*, *port*, *type*, *related_address=None*, *related_port=None*, *tcptype=None*, *generation=None*)

An ICE candidate.

classmethod from_sdp(*sdp*)

Parse a *Candidate* from SDP.

```
Candidate.from_sdp(
 '6815297761 1 udp 659136 1.2.3.4 31102 typ host generation 0')
```

to_sdp()

Return a string representation suitable for SDP.

Return type
str

CHAPTER

TWO

CHANGELOG

2.1 0.7.7

- Close underlying transport when a TURN allocation is deleted.
- Shutdown mDNS stack when it is no longer referenced.
- Rewrite asynchronous tests as coroutines.

2.2 0.7.6

- Ensure *dnspython* version is at least 2.0.0.
- Avoid 400 error when using TURN with concurrent sends.
- Avoid error if a connection is lost before the local candidate is set.

2.3 0.7.5

- Emit an event when the ICE connection is closed.

2.4 0.7.4

- Perform mDNS resolution using an A query, as Firefox does not respond to ANY queries.
- Use full module name to name loggers.

2.5 0.7.3

- Defer mDNS lock initialisation to avoid mismatched event-loops, fixes errors seen with uvloop.
- Correctly gather STUN candidates when there are multiple components.

2.6 0.7.2

- Add support for resolving mDNS candidates.

2.7 0.7.1

2.7.1 TURN

- Use the LIFETIME attribute returned by the server to determine the time-to-expiry for the allocation.
- Raise stun.TransactionFailed if TURN allocation request is rejected with an error.
- Handle 438 (Stale Nonce) error responses.
- Ignore STUN transaction errors when deleting TURN allocation.
- Periodically refresh channel bindings.

2.8 0.7.0

2.8.1 Breaking

- Make `Connection.add_remote_candidate()` a coroutine.
- Remove the *Connection.remote_candidates* setter.

CHAPTER

THREE

LICENSE

PYTHON MODULE INDEX

a

INDEX