pytgvoip Documentation

Release 0.0.6

bakatrouble

Feb 07, 2020

Contents

Telegram VoIP Library for Python

Community

PytgVoIP is a Telegram VoIP library written in Python and C++.

It uses libtgvoip (a library used in official clients) for voice encoding and transmission, and pybind11 for simple generation of Python extension written in C++.

CHAPTER 1

Features

- Making and receiving Telegram calls
- Python callbacks for sending and receiving audio stream frames allow flexible control
- Pre-built Windows wheels in PyPI

CHAPTER 2

Requirements

- Python 3.4 or higher
- An MTProto client (i.e. Pyrogram, Telethon)

CHAPTER 3

Installing

Refer the corresponding section: *Installation*

CHAPTER 4

Encoding audio streams

Streams consumed by `libtgvoip` should be encoded in 16-bit signed PCM audio.

```
$ ffmpeg -i input.mp3 -f s16le -ac 1 -ar 48000 -acodec pcm_s16le input.raw  # encode
$ ffmpeg -f s16le -ac 1 -ar 48000 -acodec pcm_s16le -i output.raw output.mp3  # decode
```

CHAPTER 5

Copyright & License

- Copyright (C) 2019 bakatrouble
- Licensed under the terms of the GNU Lesser General Public License v3 or later (LGPLv3+)

5.1 Installation

5.1.1 Requirements

On Linux and macOS to install this library you must have `make`, `cmake`, C++11 compatible compiler, Python headers, Opus and OpenSSL libraries and headers installed:

- Debian-based distributions

```
$ apt install make cmake gcc g++ python3-dev gcc g++ openssl libssl-dev libopus0 libopus-dev
```

- Archlinux-based distributions

```
$ pacman -S make cmake gcc python3 openssl opus
```

- macOS

```
$ brew install make cmake gcc g++ python3 openssl opus
```

5.1.2 Install pytgvoip

- Stable version:

```
$ pip install pytgvoip
```

- Development version:

```
$ pip install git+https://github.com/bakatrouble/pytgvoip#egg=pytgvoip
```

5.2 Usage

5.2.1 Common

- You should have a protocol object: `phoneCallProtocol(min_layer=65, max_layer=VoIPController.CONNECTION_MAX_LAYER, udp_p2p=True, udp_reflector=True)`
- All VoIP-related updates have type of `updatePhoneCall` with `phone_call` field of types `phoneCallEmpty`, `phoneCallWaiting`, `phoneCallRequested`, `phoneCallAccepted`, `phoneCall` or `phoneCallDiscarded`
- Use *`tgvoip.utils.generate_visualization()`* with `auth_key` and `g_a` for outgoing or `g_a_or_b` for incoming calls to get emojis if you need them

5.2.2 Starting conversation

- Create a `VoIPController` instance
- Call *`tgvoip.VoIPController.set_send_audio_frame_callback()`* (see docs for arguments) if needed, otherwise silence will be sent
- Call *`tgvoip.VoIPController.set_recv_audio_frame_callback()`* (see docs for arguments) if needed, otherwise nothing will be done to incoming audio stream
- Add state change handlers to *`tgvoip.VoIPController.call_state_changed_handlers`* (see docs for handler format) list if needed
- Add signal bars change handlers to *`tgvoip.VoIPController.signal_bars_changed_handlers`* (see docs for handler format) list if needed
- Invoke `help.getConfig()` (result is later referred as `config`)
- Call *`tgvoip.VoIPController.set_config()`* (arguments are: `config.call_packet_timeout_ms / 1000., config.call_connect_timeout_ms / 1000., DataSaving.NEVER, call.id`)
- Call *`tgvoip.VoIPController.set_encryption_key()`* (arguments are: `i2b(auth_key), is_outgoing` where `is_outgoing` is a corresponding boolean value)
- Build a `list` of `tgvoip.Endpoint` objects from `call.connection` (single) and `call.alternative_connections` (another list)
- Call *`tgvoip.VoIPController.set_remote_endpoints()`* (arguments are: `endpoints, call.p2p_allowed, False, call.protocol.max_layer`)
- Call *`tgvoip.VoIPController.start()`*
- Call *`tgvoip.VoIPController.connect()`*

5.2.3 Discarding call

- Build `peer`: `inputPhoneCall(id=call.id, access_hash=call.access_hash)`

- Get call duration using `tgvoip.VoIPController.call_duration`
- Get connection ID using `tgvoip.VoIPController.get_preferred_relay_id()`
- Build a suitable `reason` object (types are: `phoneCallDiscardReasonBusy`, `phoneCallDiscardReasonDisconnect`, `phoneCallDiscardReasonHangup`, `phoneCallDiscardReasonMissed`)
- Invoke `phone.discardCall(peer, duration, connection_id, reason)`. You might get `CALL_ALREADY_DECLINED` error, this is fine
- Destroy the `tgvoip.VoIPController` object

5.2.4 Ending conversation

- Send call rating and debug log if call ended normally (not failed): *TBD*
- Destroy the `tgvoip.VoIPController` object, everything will be done automatically

5.2.5 Making outgoing calls

- Get a `user_id` object for user you want to call (of type `inputPeerUser`)
- Request a Diffie-Hellman config using `messages.getDhConfig(version=0, random_length=256)`
- Check received config using `tgvoip.utils.check_dhc()`. If check is not passed, do not make the call. You might want to cache received config because check is expensive
- Choose a random value `a`, `1 < a < dhc.p-1`
- Calculate `g_a`: `pow(dhc.g, a, dhc.p)`
- Calculate `g_a_hash`: `sha256(g_a)`
- Choose a random value `random_id`, `0 <= random_id <= 0x7fffffff-1`
- Invoke `phone.requestCall(user_id, random_id, g_a_hash, protocol)`
- Wait for an update with `phoneCallAccepted` object, it means that other party has accepted the call. You also might get a `phoneCallDiscarded` object, it means that other party has declined the call
- If you have got a `phoneCallDiscarded` object, stop the `tgvoip.VoIPController`. Otherwise, continue
- Check a `g_b` value from received `phoneCallAccepted` (later referred as `call`) object using `tgvoip.utils.check_g()`. If check is not passed, stop the call
- Calculate `auth_key`: `pow(call.g_b, a, dhc.p)`
- Calculate `key_fingerprint` using `tgvoip.utils.calc_fingerprint()`
- Build `peer`: `inputPhoneCall(id=call.id, access_hash=call.access_hash)`
- Invoke `phone.confirmCall(key_fingerprint, peer, g_a, protocol)`
- Start the conversation

5.2.6 Receiving calls

- You will receive an update containing `phoneCallRequested` object (later referred as `call`). You might discard it right away (use `0` for duration and connection_id)
- Request a Diffie-Hellman config using `messages.getDhConfig(version=0, random_length=256)`
- Check received config using *`tgvoip.utils.check_dhc()`*. If check is not passed, do not make the call. You might want to cache received config because check is expensive
- Choose a random value `b`, `1 < b < dhc.p-1`
- Calculate `g_b`: `pow(dhc.g, b, dhc.p)`
- Save `call.g_a_hash`
- Build `peer`: `inputPhoneCall(id=call.id, access_hash=call.access_hash)`
- Invoke `phone.acceptCall(peer, g_b, protocol)`. You might get `CALL_ALREADY_DISCARDED` or `CALL_ALREADY_ACCEPTED` errors, then you should stop current conversation. Also, if response contains `phoneCallDiscarded` object you should stop the call
- Wait for an update with `phoneCall` object (later referred as `call`)
- Check that `call.g_a_or_b` is not empty and `sha256(call.g_a_or_b)` equals to `g_a_hash` you saved before. If it doesn't match, stop the call
- Check a `call.g_a_or_b` value object using *`tgvoip.utils.check_g()`* (second argument is `dhc.p`). If check is not passed, stop the call
- Calculate `auth_key`: `pow(call.g_a_or_b, b, dhc.p)`
- Calculate `key_fingerprint` using *`tgvoip.utils.calc_fingerprint()`*
- Check that `key_fingerprint` you have just calculated matches `call.key_fingerprint`. If it doesn't match, stop the call
- Start the conversation

5.3 libtgvoip wrapper

5.3.1 VoIPController

class `tgvoip.`**`VoIPController`**(*persistent_state_file: str = ''*, *debug=False*, *logs_dir='logs'*)

A wrapper around C++ wrapper for libtgvoip `VoIPController`

Parameters

- **`persistent_state_file`** (`str`, *optional*) – ?, empty to not use
- **`debug`** (`bool`, *optional*) – Modifies logging behavior
- **`logs_dir`** (`str`, *optional*) – Logs directory

Class attributes:

LIBTGVOIP_VERSION Used `libtgvoip` version

CONNECTION_MAX_LAYER Maximum layer supported by used `libtgvoip` version

persistent_state_file
Value set in the constructor

call_state_changed_handlers
`list` of call state change callbacks, callbacks receive a *CallState* object as argument

signal_bars_changed_handlers
`list` of signal bars count change callbacks, callbacks receive an `int` object as argument

call_duration
Current call duration in seconds as `int` if call was started, otherwise 0

start()
Start the controller

connect()
Start the call

set_proxy(*address: str*, *port: int = 1080*, *username: str = ”*, *password: str = ”*)
Set SOCKS5 proxy config

Parameters

- **address** (`str`) – Proxy hostname or IP address
- **port** (`int`, *optional*) – Proxy port
- **username** (`int`, *optional*) – Proxy username
- **password** (`int`, *optional*) – Proxy password

Raises `ValueError` if `address` is empty

set_encryption_key(*key: bytes*, *is_outgoing: bool*)
Set call auth key

Parameters

- **key** (`bytes`) – Auth key, must be exactly 256 bytes
- **is_outgoing** (`bool`) – Is call outgoing

Raises `ValueError` if provided auth key has wrong length

set_remote_endpoints(*endpoints: List[<sphinx.ext.autodoc.importer._MockObject object at 0x7f93565b59e8>], allow_p2p: bool, tcp: bool, connection_max_layer: int*)
Set remote endpoints received in call object from Telegram.

Usually it’s `[call.connection] + call.alternative_connections`.

You must build `Endpoint` objects from MTProto `phoneConnection` objects and pass them in list.

Parameters

- **endpoints** (`list` of `Endpoint`) – List of endpoints
- **allow_p2p** (`bool`) – Is p2p connection allowed, usually *call.p2p_allowed* value is used
- **tcp** (`bool`) – Connect via TCP, not recommended
- **connection_max_layer** (`int`) – Use a value provided by `VoIPController.CONNECTION_MAX_LAYER`

Raises `ValueError` if either no endpoints are provided or endpoints without IPv4 or with wrong `peer_tag` (must be either `None` or have length of 16 bytes) are detected

get_debug_string() → str
Get debug string

Returns: `str` containing debug info

set_network_type(*_type: tgvoip.tgvoip.NetType*)
Set network type

Parameters **_type** (*NetType*) – Network type to set

set_mic_mute(*mute: bool*)
Set "microphone" state. If muted, audio is not being sent

Parameters **mute** (`bool`) – Whether to mute "microphone"

set_config(*recv_timeout: float*, *init_timeout: float*, *data_saving_mode: tgvoip.tgvoip.DataSaving*, *call_id: int*, *enable_aec: bool = True*, *enable_ns: bool = True*, *enable_agc: bool = True*, *log_file_path: str = None*, *status_dump_path: str = None*, *log_packet_stats: bool = None*)
Set call config

Parameters

- **recv_timeout** (`float`) – Packet receive timeout, usually value received from `help.getConfig()` is used
- **init_timeout** (`float`) – Packet init timeout, usually value received from `help.getConfig()` is used
- **data_saving_mode** (*DataSaving*) – Data saving mode
- **call_id** (`int`) – Call ID
- **enable_aec** (`bool`, *optional*) – Whether to enable automatic echo cancellation, defaults to `True`
- **enable_ns** (`bool`, *optional*) – Whether to enable noise suppression, defaults to `True`
- **enable_agc** (`bool`, *optional*) – Whether to enable automatic gain control, defaults to `True`
- **log_file_path** (`str`, *optional*) – Call log file path, calculated automatically if not provided
- **status_dump_path** (`str`, *optional*) – Status dump path, calculated automatically if not provided and `debug` is enabled
- **log_packet_stats** (`bool`, *optional*) – Whether to log packet stats, defaults to `debug` value

debug_ctl(*request: int*, *param: int*)
Debugging options

Parameters

- **request** (`int`) – Option (`1` for max bitrate, `2` for packet loss (in percents), `3` for toggling p2p, `4` for toggling echo cancelling)
- **param** (`int`) – Numeric value for options 1 and 2, `0` or `1` for options 3 and 4

get_preferred_relay_id() → int
Get preferred relay ID (used in `discardCall` MTProto request)

Returns `int` ID

get_last_error() → tgvoip.tgvoip.CallError
Get last error type

Returns *CallError* matching last occurred error type

get_stats() → <sphinx.ext.autodoc.importer._MockObject object at 0x7f93565b56d8>
Get call stats

Returns `Stats` object

get_debug_log() → str
Get debug log

Returns JSON `str` containing debug log

set_audio_output_gain_control_enabled(*enabled: bool*)
Toggle output gain control

Parameters **enabled** (`bool`) – Whether to enable output gain control

set_echo_cancellation_strength(*strength: int*)
Set echo cancellation strength, does nothing currently but was in Java bindings (?)

Parameters **strength** (`int`) – Strength value

get_peer_capabilities() → int
Get peer capabilities

Returns `int` with bit mask, looks like it is used only for experimental features (group, video calls)

need_rate() → bool
Get whether the call needs rating

Returns `bool` value

update_state(*state: tgvoip.tgvoip.CallState*)
Manually update state (only triggers handlers)

Parameters **state** (*CallState*) – State to set

set_send_audio_frame_callback(*func: callable*)
Set callback providing audio data to send

Should accept one argument (`int` length of requested audio frame) and return `bytes` object with audio data encoded in 16-bit signed PCM

If returned object has insufficient length, it will be automatically padded with zero bytes

Parameters **func** (`callable`) – Callback function

set_recv_audio_frame_callback(*func: callable*)
Set callback receiving incoming audio data

Should accept one argument (`bytes`) with audio data encoded in 16-bit signed PCM

Parameters **func** (`callable`) – Callback function

5.3.2 VoIPServerConfig

class tgvoip.**VoIPServerConfig**(**args*, ***kwargs*)
Global server config class. This class contains default config in its source

classmethod set_config(*_json: Union[str, dict]*)
Set global server config

Parameters **_json** (str | dict) – either JSON-encoded object or dict containing config values. Might be received from MTProto phone.getCallConfig() call, if not set default values are used

Raises Prints an error to stderr if JSON parsing (for str argument) or encoding (for dict argument) has occurred

classmethod set_bitrate_config(*init_bitrate: int = 16000, max_bitrate: int = 20000, min_bitrate: int = 8000, decrease_step: int = 1000, increase_step: int = 1000*)

Helper method for setting bitrate options

Parameters

- **init_bitrate** (int) – Initial bitrate value
- **max_bitrate** (int) – Maximum bitrate value
- **min_bitrate** (int) – Minimum bitrate value
- **decrease_step** (int) – Bitrate decrease step
- **increase_step** (int) – Bitrate increase step

Raises Same as *set_config()*

5.3.3 Enums

class tgvoip.**NetType**

An enumeration of network types

Members:

- UNKNOWN = 0
- GPRS = 1
- EDGE = 2
- NET_3G = 3
- HSPA = 4
- LTE = 5
- WIFI = 6
- ETHERNET = 7
- OTHER_HIGH_SPEED = 8
- OTHER_LOW_SPEED = 9
- DIALUP = 10
- OTHER_MOBILE = 11

class tgvoip.**DataSaving**

An enumeration of data saving modes

Members:

- NEVER = 0
- MOBILE = 1
- ALWAYS = 2

class tgvoip.**CallState**
An enumeration of call states

Members:

- WAIT_INIT = 1
- WAIT_INIT_ACK = 2
- ESTABLISHED = 3
- FAILED = 4
- RECONNECTING = 5
- HANGING_UP = 10
- ENDED = 11
- EXCHANGING_KEYS = 12
- WAITING = 13
- REQUESTING = 14
- WAITING_INCOMING = 15
- RINGING = 16
- BUSY = 17

class tgvoip.**CallError**
An enumeration of call errors

Members:

- UNKNOWN = 0
- INCOMPATIBLE = 1
- TIMEOUT = 2
- AUDIO_IO = 3
- PROXY = 4

5.3.4 Data structures

class tgvoip.**Stats**
Object storing call stats

bytes_sent_wifi
Amount of data sent over WiFi :type: `int`

bytes_sent_mobile
Amount of data sent over mobile network :type: `int`

bytes_recvd_wifi
Amount of data received over WiFi :type: `int`

bytes_recvd_mobile
Amount of data received over mobile network :type: `int`

class tgvoip.**Endpoint**
Object storing endpoint info

Parameters

- **_id** (int) – Endpoint ID
- **ip** (str) – Endpoint IPv4 address
- **ipv6** (str) – Endpoint IPv6 address
- **port** (int) – Endpoint port
- **peer_tag** (bytes) – Endpoint peer tag

5.4 Utility functions

`tgvoip.utils.`**`i2b`**(*value: int*) → bytes
Convert integer value to bytes

Parameters **value** (int) – Value to convert

Returns Resulting bytes object

`tgvoip.utils.`**`b2i`**(*value: bytes*) → int
Convert bytes value to integer

Parameters **value** (bytes) – Value to convert

Returns Resulting int object

`tgvoip.utils.`**`check_dhc`**(*g: int*, *p: int*) → None
Security checks for Diffie-Hellman prime and generator. Ported from Java implementation for Android

Parameters

- **g** (int) – DH generator
- **p** (int) – DH prime

Raises ValueError if checks are not passed

`tgvoip.utils.`**`check_g`**(*g_x: int*, *p: int*) → None
Check g_ numbers

Parameters

- **g_x** – g_ number to check
- **p** – DH prime

Raises ValueError if checks are not passed

`tgvoip.utils.`**`calc_fingerprint`**(*key: bytes*) → int
Calculate key fingerprint

Parameters **key** (bytes) – Key to generate fingerprint for

Returns int object representing a key fingerprint

`tgvoip.utils.`**`generate_visualization`**(*key: Union[bytes, int], part2: Union[bytes, int]) -> (typing.List[str], typing.List[str]*)
Generate emoji visualization of key

https://core.telegram.org/api/end-to-end/voice-calls#key-verification

Parameters

- **key** (bytes | int) – Call auth key
- **part2** (bytes | int) – *g_a* value of the caller

Returns A tuple containing two lists (of emoji strings and of their text representations)

`tgvoip.utils.`**`get_real_elapsed_time`**`()` → float

Get current performance counter value

Returns Time to use for measuring call duration

Python Module Index

t

Index

T

U

V