Haka MQTT Documentation

Release 0.3.5

Keegan Callin

Feb 13, 2019

Contents:

The *haka_mqtt* package is reliable "weapons grade" MQTT client library. It contains a core mqtt reactor class built with provable reliability, and reproducibility as its fundamental goals. It turns out that it is sufficiently speedy as well; there has never been a performance complaint lodged.

CHAPTER 1

Status

The project's core reactor is stable. It has been tested on systems with thousands of distributed nodes in difficult field conditions. The QoS=1 datapath is well field tested. The QoS=0 and QoS=2 are not as thoroughly field tested.

While the core reactor is very well tested the frontends are less tested. You should pay attention to notes on the different frontends regarding their status and use.

The haka library is mostly tested on Linux derivatives. It may work on other platforms but this has not been tested by the authors and no definite reports of success have been reported to the authors.

CHAPTER 2

Installation

The haka-mqtt package is distributed through pypi.org and can be installed with the standard Python package manager pip:

```
$ pip install haka-mqtt
```

If you do not have pip then the package can be downloaded from haka-mqtt and installed with the standard *setup.py* method:

```
$ python setup.py install
```

CHAPTER 3

Project Infrastructure

The project is coordinated through public infrastructure available at several places:

- Releases (pypi)
- Documentation (readthedocs.io)
- Bug Tracker (github)
- Code Repository (github)

CHAPTER 4

Table of Contents

4.1 User Guide

The *haka_mqtt* package is reliable "weapons grade" MQTT client library. It contains a core mqtt reactor class built with provable reliability, and reproducibility as its fundamental goals. A side effect is that the library turns out to be speedy as well.

The core reactor takes some time to plumb into an application event loop. To make life easier for simple use cases *haka-mqtt* includes a number of front-ends that speed implementation by making some reasonable assumptions. If these assumptions do not hold for application then it best to use the core reactor directly.

4.1.1 Quickstart

Installation through pip is supported:

```
pip install haka-mqtt
```

A basic mqtt client is short and sweet:

```python
"""Basic client that connects to test.mosquitto.org, subscribes
to a topic, publishes to that topic, awaits notification of
publish on that topic, then cleanly disconnects."""

# Standard python Packages
import logging

# 3rd-Party Packages
from haka_mqtt.frontends.poll import (
    MqttPollClientProperties,
    BlockingMqttClient
)
from haka_mqtt.reactor import ACTIVE_STATES
from mqtt_codec.packet import MqttTopic
```

(continues on next page)

(continued from previous page)

```
LOG_FMT='%(asctime)s %(name)s %(levelname)s %(message)s'
logging.basicConfig(format=LOG_FMT, level=logging.INFO)

properties = MqttPollClientProperties()
properties.host = 'test.mosquitto.org'
properties.port = 1883
properties.ssl = False

TOPIC = 'haka'

c = BlockingMqttClient(properties)
c.start()
sub_ticket = c.subscribe([MqttTopic(TOPIC, 1)])
c.on_suback = lambda c, suback: c.publish(TOPIC, 'payload', 1)
c.on_publish = lambda c, publish: c.stop()

while c.is_active():
    c.poll(5.)
```

Typical output of this program is shown:

```
2018-11-05 22:30:41,655 haka INFO Starting.
2018-11-05 22:30:41,655 haka INFO Looking up host test.mosquitto.org:1883.
2018-11-05 22:30:41,798 haka INFO Found family=inet sock=sock_stream proto=tcp␣
↪addr=37.187.106.16:1883 (chosen)
2018-11-05 22:30:41,798 haka INFO Found family=inet6 sock=sock_stream proto=tcp␣
↪addr=2001:41d0:a:3a10::1:1883
2018-11-05 22:30:41,798 haka INFO Connecting.
2018-11-05 22:30:41,798 haka INFO Connected.
2018-11-05 22:30:41,932 haka INFO Launching message MqttConnect(client_id='bobby',␣
↪clean_session=True, keep_alive=0s, username=***, password=***, will=None).
2018-11-05 22:30:41,933 haka INFO Launching message MqttSubscribe(packet_id=0,␣
↪topics=[Topic('haka', max_qos=1)]).
2018-11-05 22:30:42,068 haka INFO Received MqttConnack(session_present=False, return_
↪code=<ConnackResult.accepted: 0>).
2018-11-05 22:30:42,225 haka INFO Received MqttSuback(packet_id=0, results=[
↪<SubscribeResult.qos1: 1>]).
2018-11-05 22:30:42,225 haka INFO Launching message MqttPublish(packet_id=1, topic=
↪'haka', payload=0x7061796c6f6164, dupe=False, qos=1, retain=False).
2018-11-05 22:30:42,376 haka INFO Received MqttPuback(packet_id=1).
2018-11-05 22:30:42,552 haka INFO Received MqttPublish(packet_id=1, topic=u'haka',␣
↪payload=0x7061796c6f6164, dupe=False, qos=1, retain=False).
2018-11-05 22:30:42,552 haka INFO Stopping.
2018-11-05 22:30:42,552 haka INFO Launching message MqttDisconnect().
2018-11-05 22:30:42,552 haka INFO Shutting down outgoing stream.
2018-11-05 22:30:42,686 haka INFO Remote has gracefully closed remote->local writes;␣
↪Stopped.
```

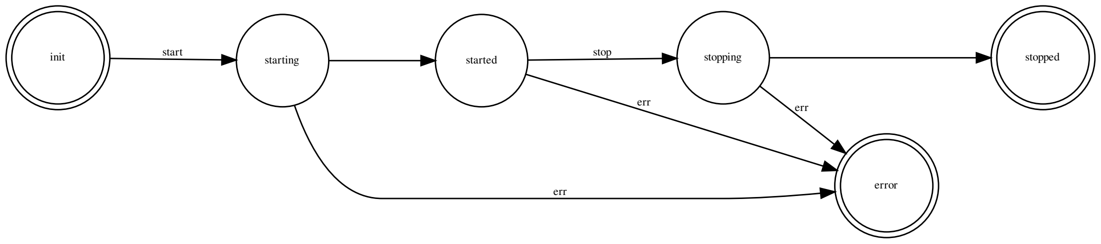
4.1.2 Core Reactor

The core MQTT Reactor is the backend behind all event loops. It is built to be used with blocking sockets or with non-blocking sockets. It does not itself integrate with any event loop and it is the different frontends that match the reactor with event loops and whatever special rule processing is required for the given application.

Reactor Lifecycle

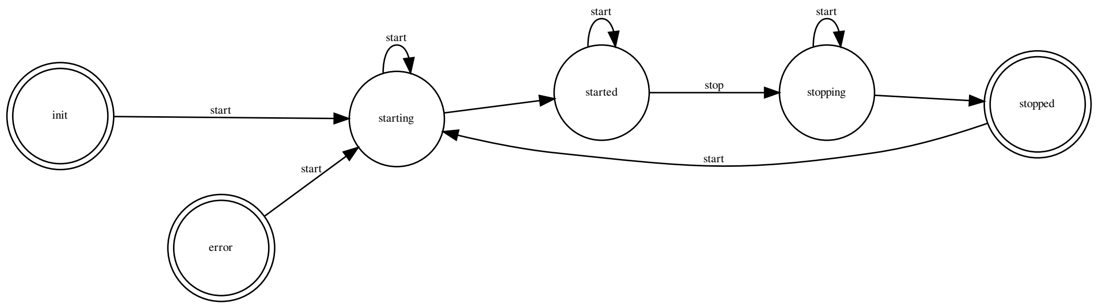
A reactor has active and inactive states. While in an inactive state no sockets are active, no DNS calls are active, and no tasks are scheduled. Any of these may be true in an active state.

The normal reactor lifecycle is summarized in this state diagram:



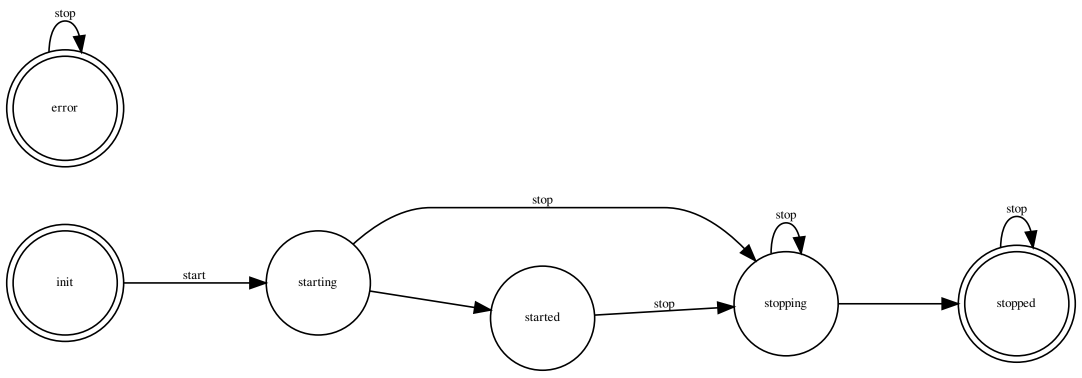
Start

Calls to `haka_mqtt.reactor.Reactor.start()` can be used to activate an inactive reactor otherwise they have no effect.



Stop

Calls to `haka_mqtt.reactor.Reactor.stop()` may be used to at the earliest possible opportunity cleanly disconnect from a server.



Subscribe/Unsubscribe

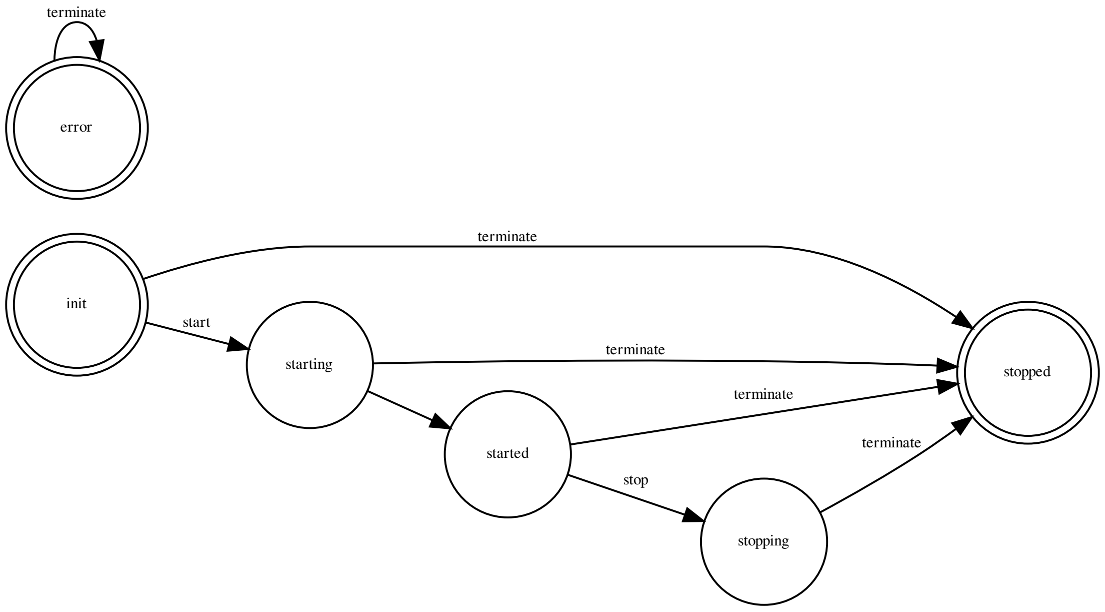
Subscribe/unsubscribe calls made before a call to *haka_mqtt.reactor.Reactor.stop()* will have their associated packets delivered before the socket outgoing write channel is closed. Whether the packets are acknowledged on not depends on server implementation.

Publish

Calls made to publish before a call to *haka_mqtt.reactor.Reactor.stop()* will have the associated packets delivered before the socket's outgoing write channel is closed. The server may or may not acknowledge QoS=1 publishes before closing the socket. QoS=2 packets may be acknowledge with a `pubrec` packet but the reactor will not acknowledge the `pubrec` packet with a `pubrel` since the outgoing socket stream would already have been closed. Any `pubrel` packets qeued before the call to stop will be delivered before the outgoing write channel is closed and may or may not be acknowledged by the server with a `pubcomp`.

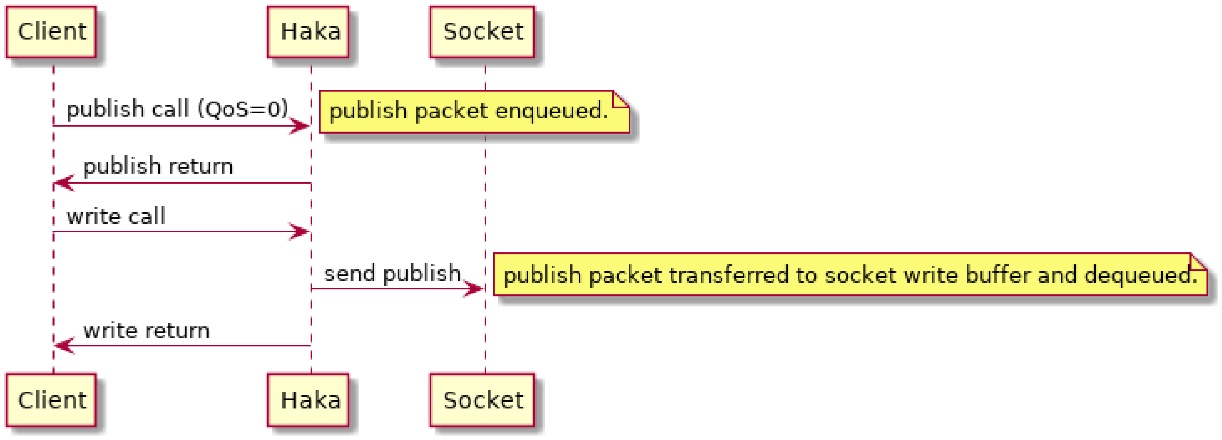
Terminate

A *haka_mqtt.reactor.Reactor.terminate()* call prompty closes all haka-mqtt reactor resources and places the reactor into a `stopped` state. All schedule deadlines are promptly cancelled. All socket resources are promptly closed. Any asynchronous hostname lookups are cancelled. "Prompt" in this case means before the `terminate` call returns.

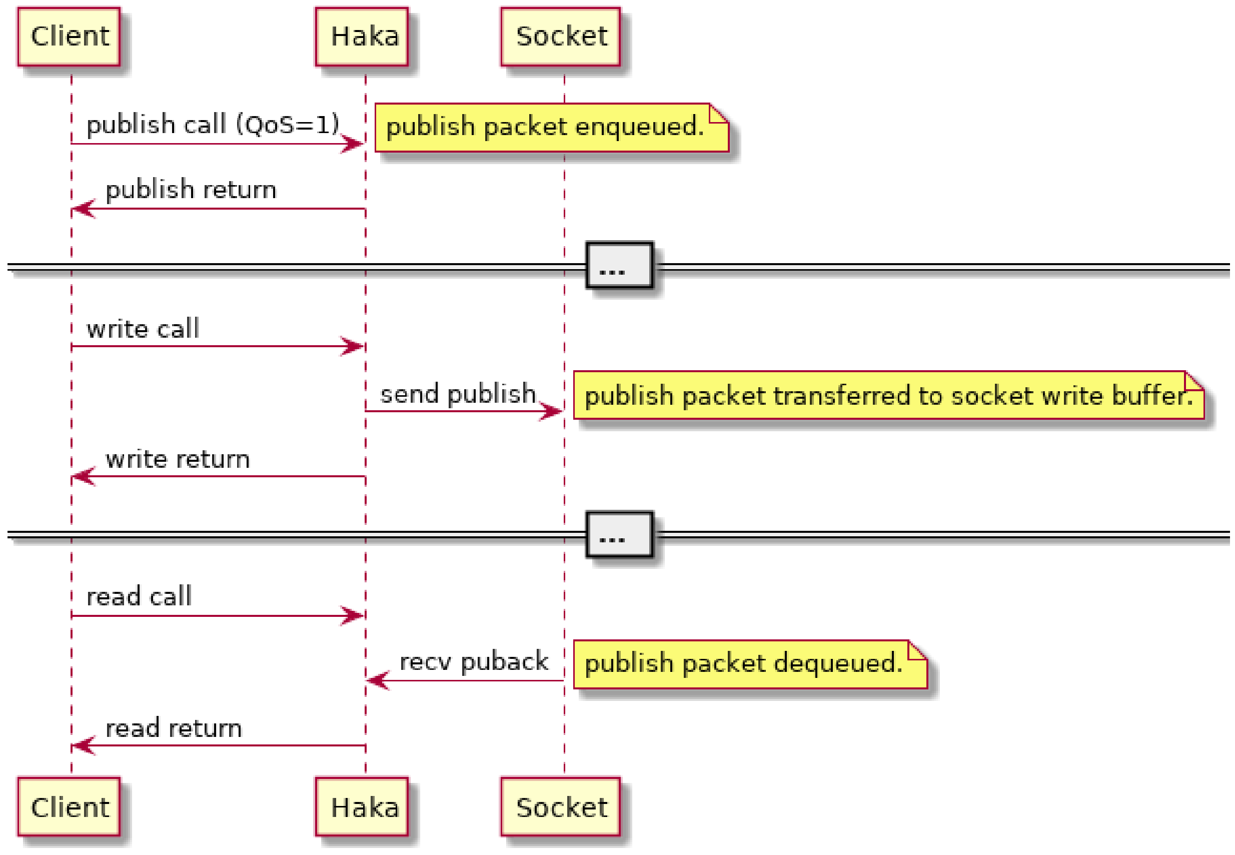


Send Path

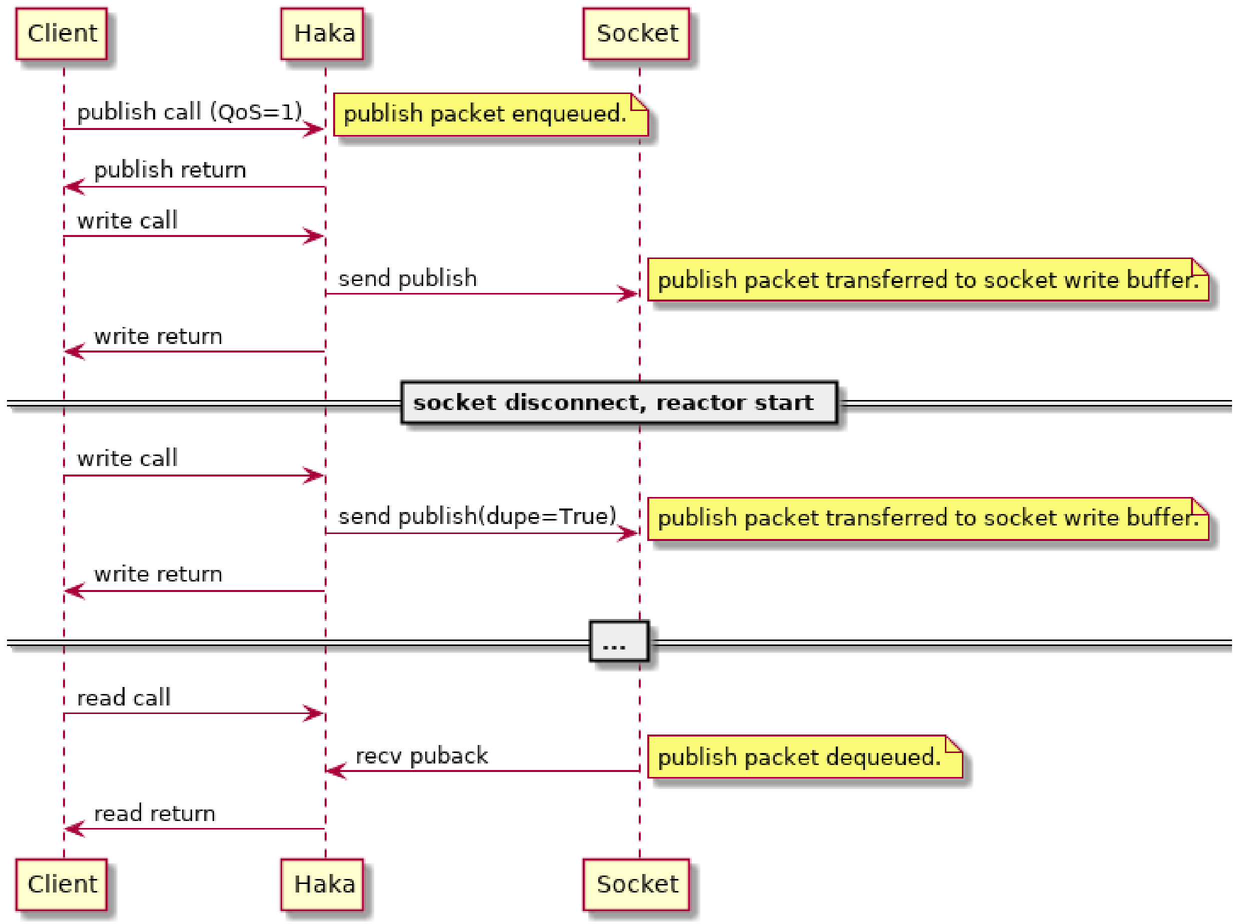
QoS 0



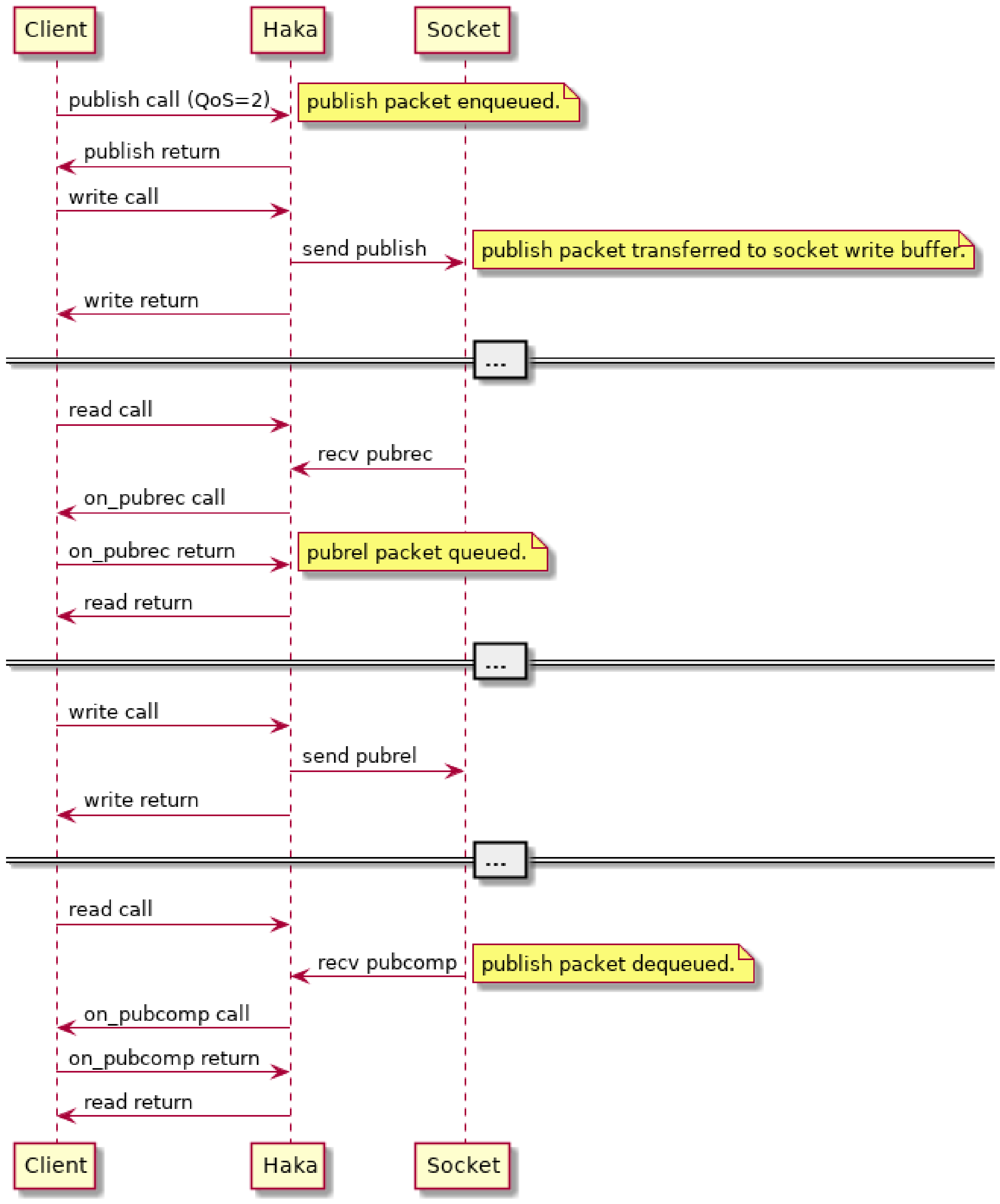
QoS 1


Client
Haka
Socket
publish call (QoS=1)
publish packet enqueued.
publish return
...
write call
send publish
publish packet transferred to socket write buffer.
write return
...
read call
recv puback
publish packet dequeued.
read return
Client
Haka
Socket

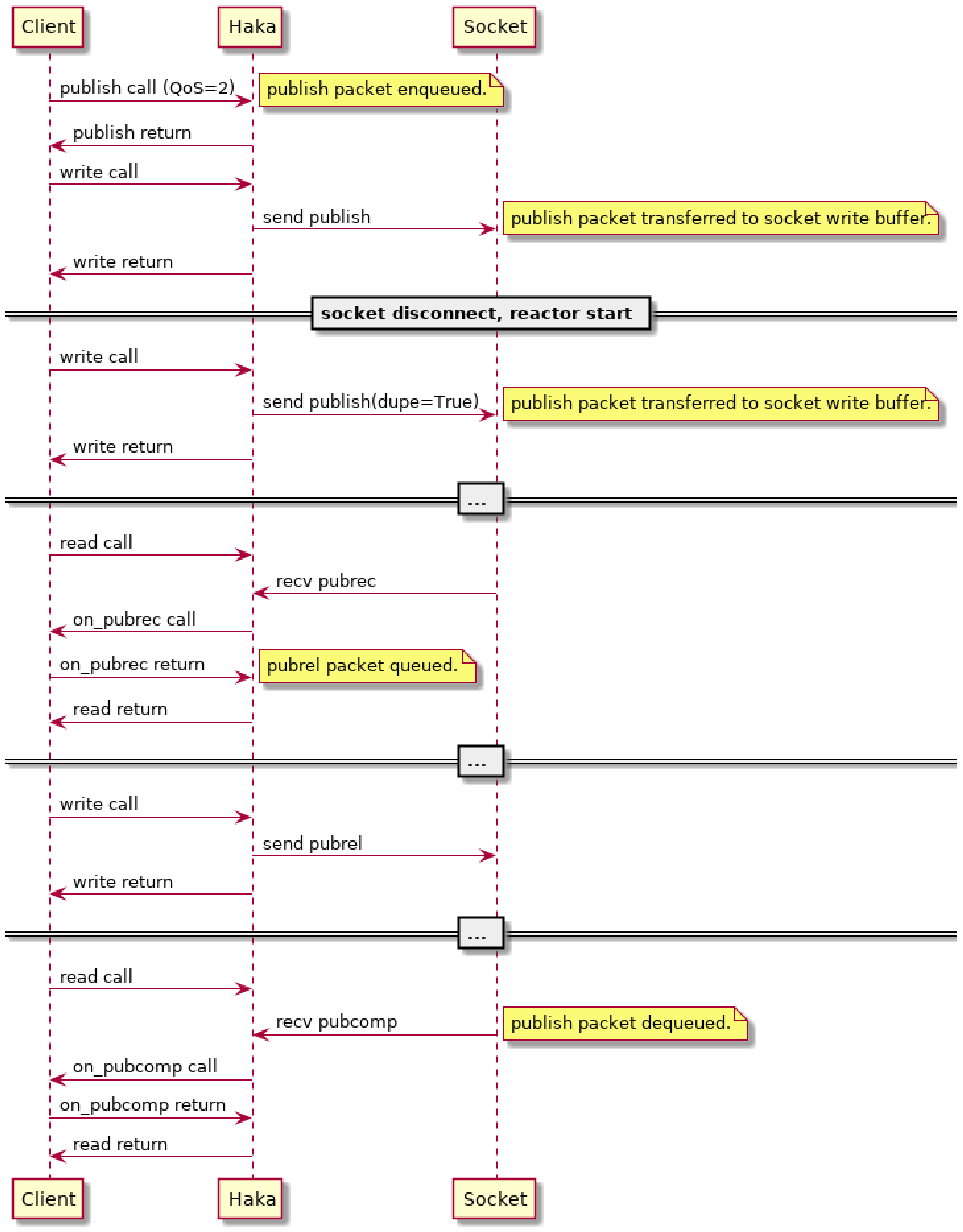
QoS 1 w/Disconnect



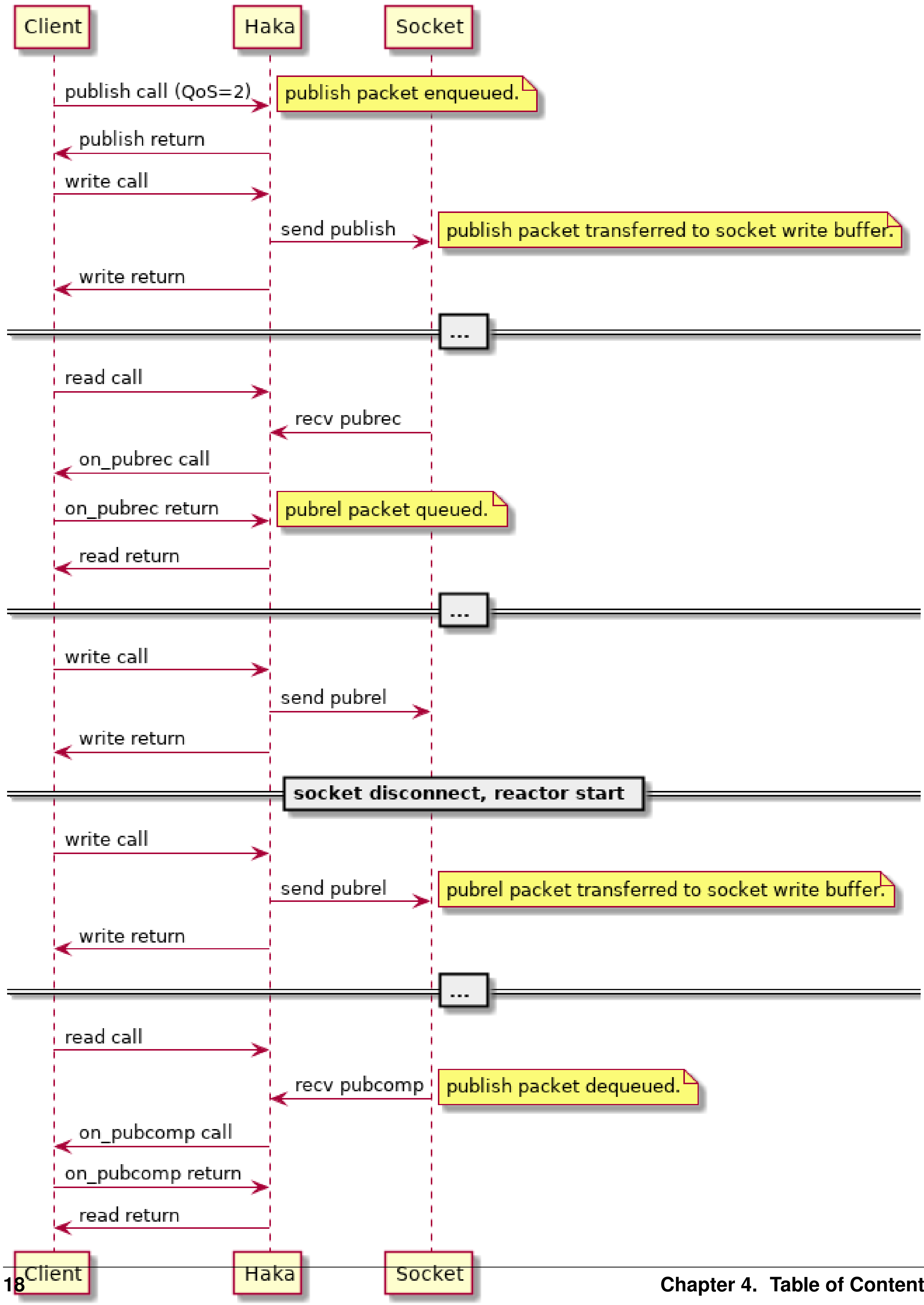
QoS 2


Client
Haka
Socket
publish call (QoS=2)
publish packet enqueued.
publish return
write call
send publish
publish packet transferred to socket write buffer.
write return
...
read call
recv pubrec
on_pubrec call
on_pubrec return
pubrel packet queued.
read return
...
write call
send pubrel
write return
...
read call
recv pubcomp
publish packet dequeued.
on_pubcomp call
on_pubcomp return
read return
Client
Haka
Socket

QoS 2 w/Publish Disconnect

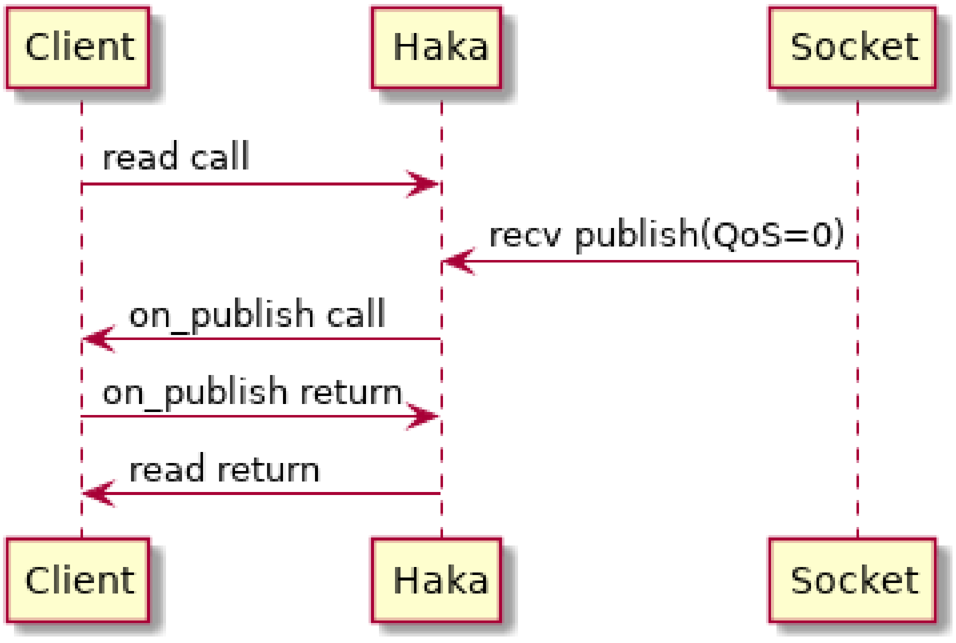


QoS 2 w/Pubrel Disconnect

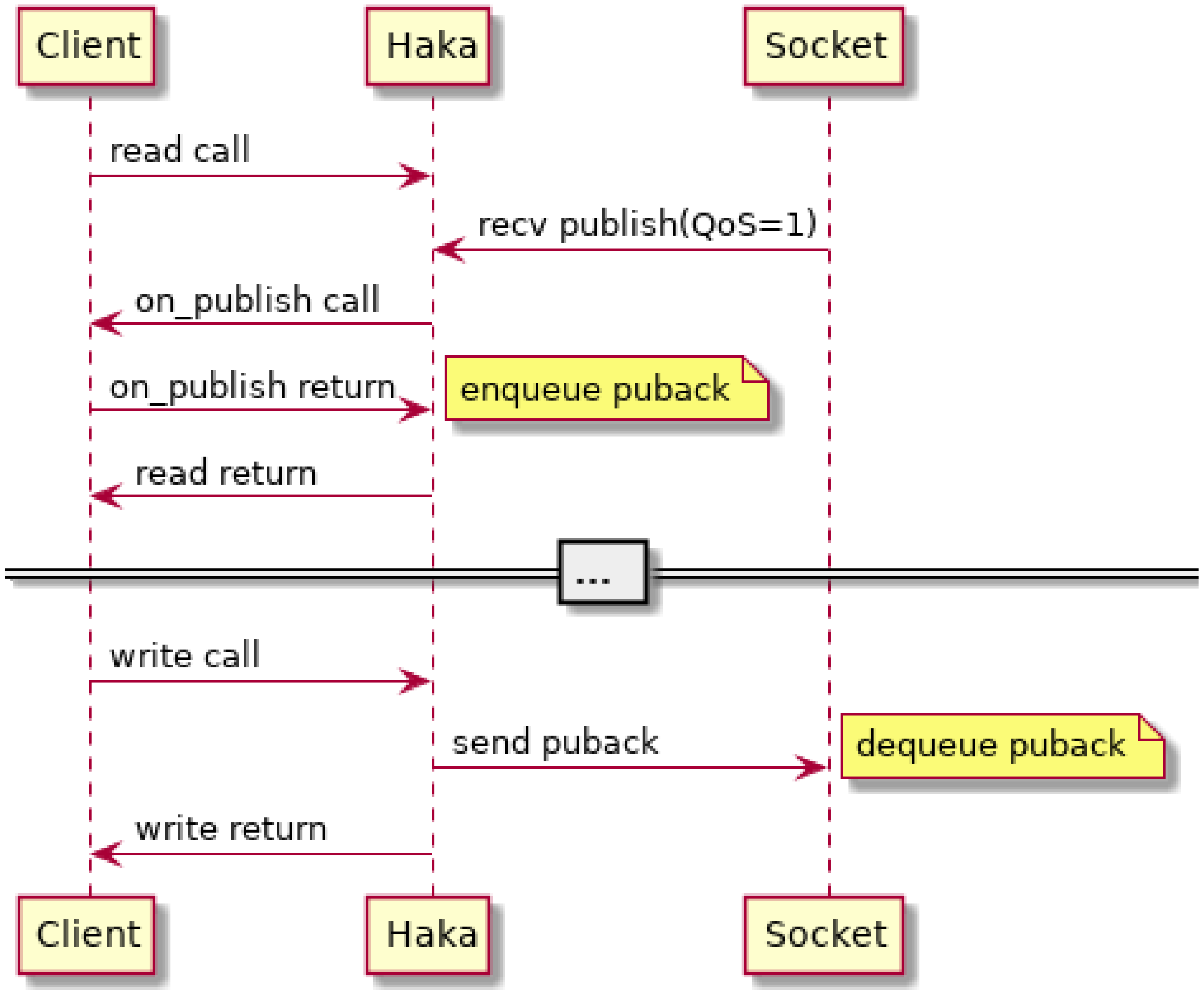


Receive Path

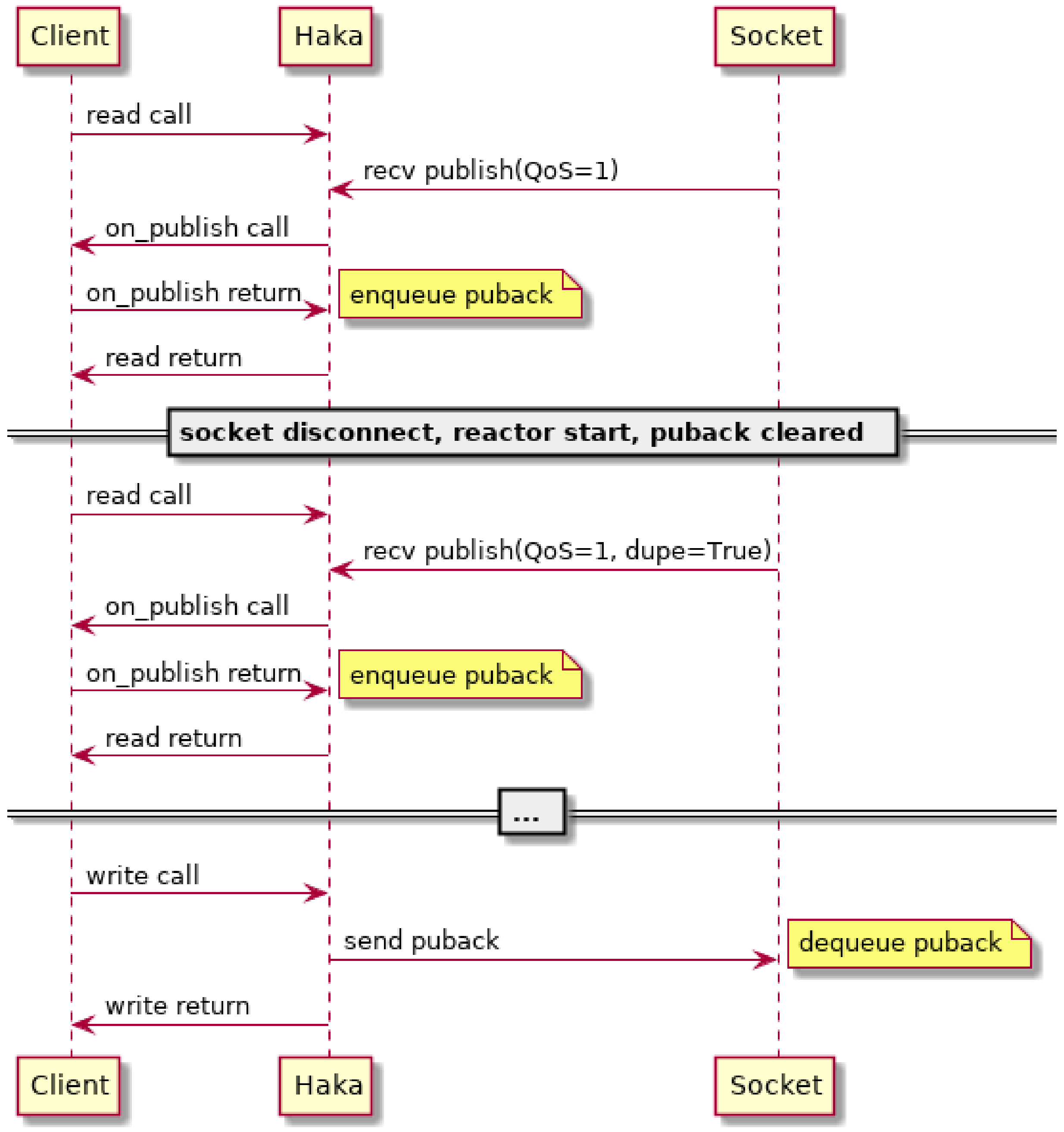
QoS 0



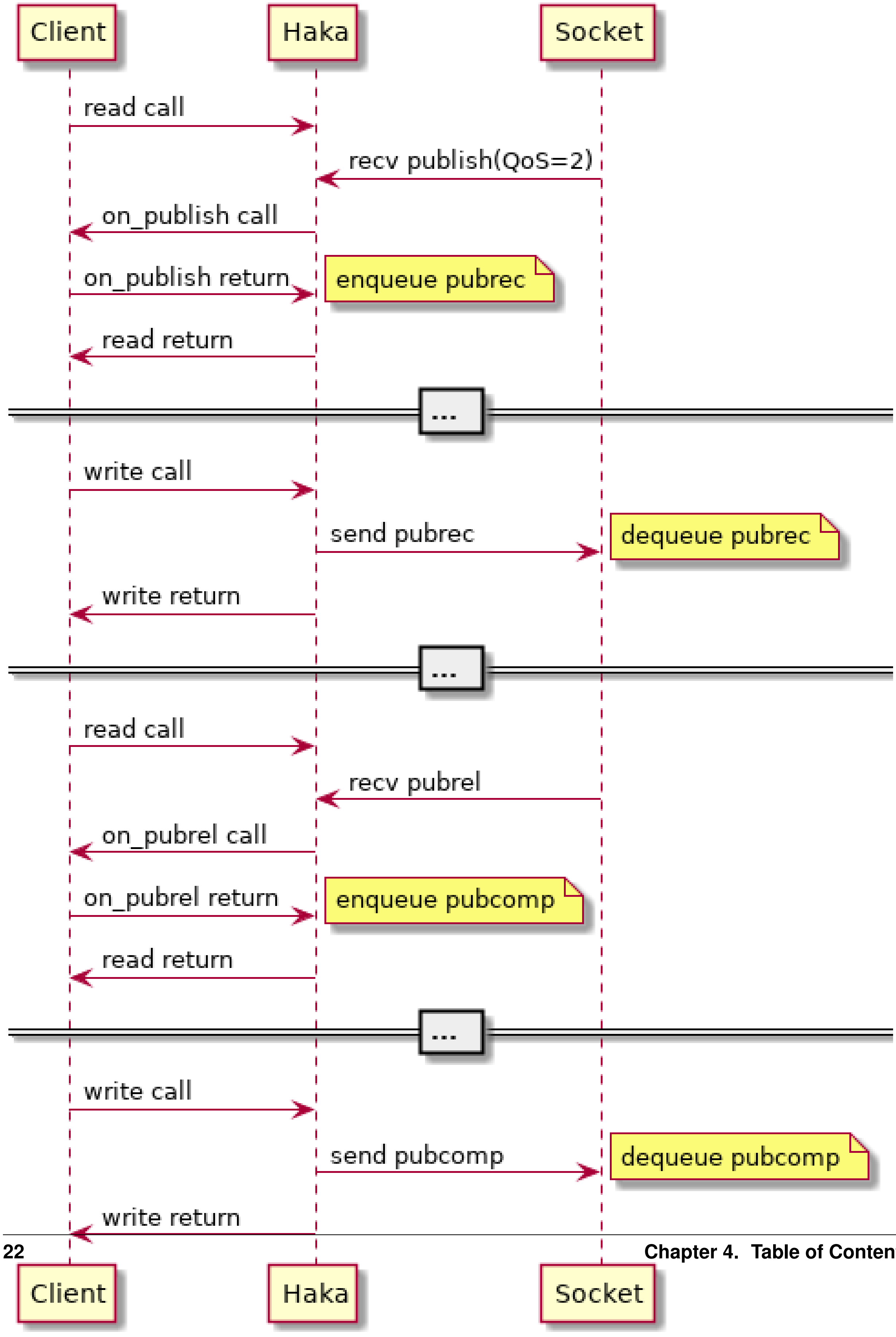
QoS 1


Client
Haka
Socket
read call
recv publish(QoS=1)
on_publish call
on_publish return
enqueue puback
read return
...
write call
send puback
dequeue puback
write return
Client
Haka
Socket

QoS 1 with Pre-Ack Disconnect



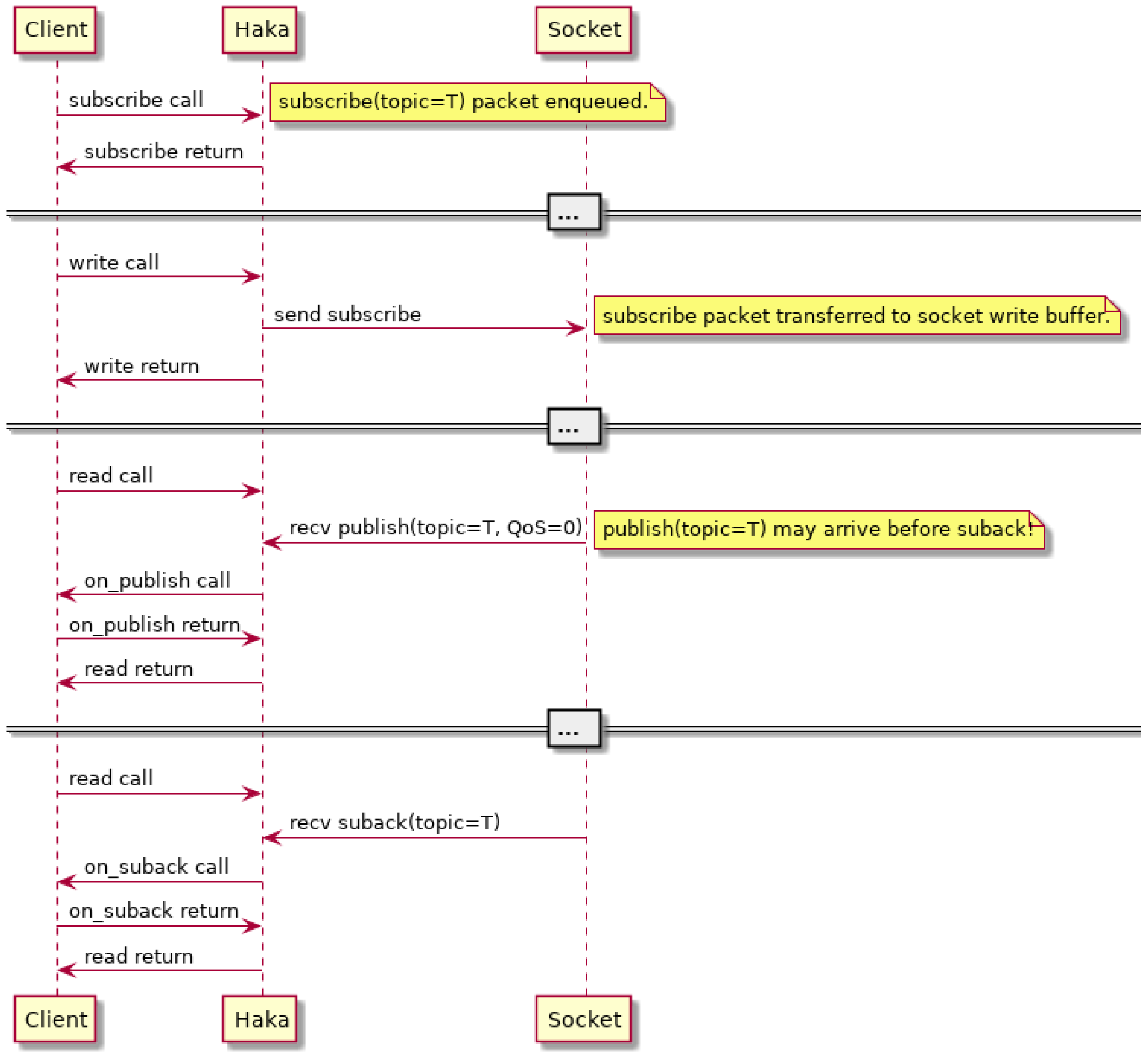
QoS 2


Client
Haka
Socket
read call
recv publish(QoS=2)
on_publish call
on_publish return
enqueue pubrec
read return
...
write call
send pubrec
dequeue pubrec
write return
...
read call
recv pubrel
on_pubrel call
on_pubrel return
enqueue pubcomp
read return
...
write call
send pubcomp
dequeue pubcomp
write return
Client
Haka
Socket

Subscribe/Unsubscribe Path

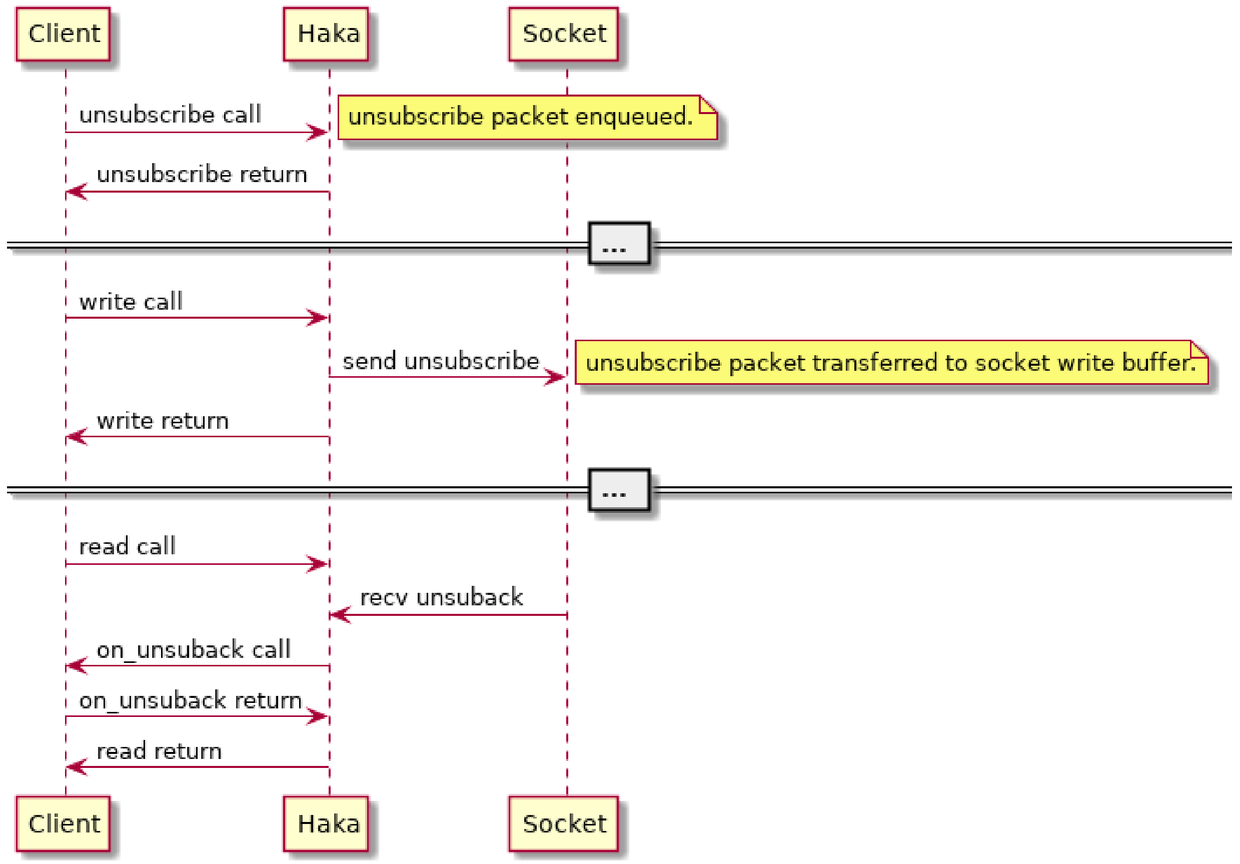
Subscribe

The subscribe/suback sequence shows the potential for a publish publish for the newly subscribed topic before the suback packet arrives.



Unsubscribe Path

The unsubscribe/unsuback sequence is very similar to the subscribe/suback sequence.



Keepalive Path

The *haka-mqtt* client can be configured to use `MqttPingreq` and `MqttPingresp` packets to test the liveliness of a connection. If the connection does not prove lively the client will disconnect from the server.

Receive Path

If the core reactor does not receive any bytes on the TCP socket for *`haka_mqtt.reactor.Reactor.recv_idle_ping_period`* seconds then a `MqttPingreq` packet will be launched. If after *`haka_mqtt.reactor.Reactor.recv_idle_abort_period`* seconds no bytes have been received then the reactor terminates the connection and enters an error state.

Send Path

If the core reactor does not send and bytes on the underlying socket for *`haka_mqtt.reactor.Reactor.keepalive_period`* seconds then a `MqttPingreq` packet will be launched. The server will disconnect the client if it does not receive any packets after 1.5 times *`haka_mqtt.reactor.Reactor.keepalive_period`* seconds [MQTT-3.1.2-24]. The client will detect this as a network disconnection.

Memory Usage

Send Path

The core MQTT Reactor wraps an outgoing message queue. Maximum memory usage should be be bounded by about 2x the byte size of the outgoing message queue.

Receive Path

The peak receive path memory usage on the order of 2x the maximum MQTT message size. In MQTT 3.1.1 the maximum message length is `mqtt_codec.packet.MqttFixedHeader.MAX_REMAINING_LEN` (268435455 bytes) so the maximum memory usage will be about 512MB. Typical MQTT messages are much smaller than this so peak memory usage will likewise be much smaller.

A possible future enhancement to the reactor could be to set a maximum receive message size lower than the protocol maximum.

Logging

By default *haka-mqtt* logs details of operational events to *haka* log through the standard Python logging framework. Alternatively custom loggers can be provided, or logging disabled entirely.

Log Levels

Notice of network failures and server protocol violations are logged at the *WARNING* level. Notice of normal operational events such as sending or receiving publish messages, connects, or normal disconnects are logged at the *INFO* level. Traces of bytes sent/received on network sockets is logged at *DEBUG* level.

Standard Python *logging* Module

By default *haka-mqtt* logs details of operational events to *haka* log through the standard Python logging framework. If a *str* is provided to the core `haka_mqtt.reactor.Reactor` log parameter then logs will be written to the logger by that name instead.

Custom Logging

If a *logging.Logger*-like class is provided to the core `haka_mqtt.reactor.Reactor` log parameter then the logger will be used as-is without a call to the standard library *logging.getLogger* method.

Disabling Logging

Logging can be disabled entirely by setting the `haka_mqtt.reactor.Reactor` log parameter to *None*.

4.1.3 Frontends

Poll/Blocking

A polling frontend is available at `haka_mqtt.frontends.poll`.

Select/epoll

The core reactor at `haka_mqtt.reactor` is suitable for use with select/epoll.

Asyncio

An asyncio-based frontend is not available at this time.

Threading

A threading-based frontend is not available at this time.

4.1.4 Examples

Since it is well reported that actions speak louder than words here are some practical worked examples.

Poll Frontend

An MQTT client that demonstrates a select-based polling interface.

A loopback client that subscribes to a topic that it publishes to. Every time a message is published it should be echoed back to the client by the remote MQTT broker.

class `examples.frontend_poll.`**`ExampleMqttClient`**(*endpoint*)

Bases: `haka_mqtt.frontends.event_queue.MqttEventEnqueue`, `haka_mqtt.frontends.poll.MqttPollClient`

A helper class for polling mqtt events.

It is critical that the order of inheritance is correct for this class to work correctly. The `MqttEventEnqueue` class *must* appear before `MqttPollClient` so that its methods are called first.

`expect_event`(*predicate*, *timeout=None*)

Waits any event to occur and returns it if `predicate(event)` returns `True`; otherwise raises an exception. If *timeout* expires before any event is received then returns *None*.

Parameters

- **`predicate`** (*`callable`*) – A callable that will be passed a single argument that will be of type `mqtt_codec.packet.MqttPacketBody` or `haka_mqtt.frontends.event_queue.MqttConnectionEvent`.
- **`timeout`** (*`float or None`*) – Maximum amount of time to wait for an event. If None then waits forever.

Raises *`UnexpectedMqttEventError`* – The first even that occurs `predicate(event)` returns False.

Returns Returns an event matching predicate(e) or None if no such event occurred before timeout.

Return type mqtt_codec.packet.MqttPacketBody or haka_mqtt.frontends.event_queue.MqttConnectionEvent or None

`poll_until_event`(*timeout=None*)

Polls connection until an event occurs then returns it. If timeout passes without an event occurring returns None.

Parameters **timeout** (*float or None*) – Maximum amount of time to wait for an event. If None then waits forever. Must satisfy condition `timeout >= 0`.

Returns None is returned if no event occurs.

Return type mqtt_codec.packet.MqttPacketBody or MqttConnectionEvent or None

exception `examples.frontend_poll.`**`UnexpectedMqttEventError`**(*e*)

Bases: `exceptions.Exception`

`examples.frontend_poll.`**`argparse_endpoint`**(*s*)

Splits an incoming string into host and port components.

```
>>> argparse_endpoint('localhost:1883')
('localhost', 1883)
```

Parameters **s** (*str*) –

Raises `ArgumentTypeError` – Raised when port number is out of range 1 <= port <= 65535, when port is not an integer, or when there is more than one colon in the string.

Returns hostname, port tuple.

Return type (str, int)

`examples.frontend_poll.`**`create_parser`**()

Creates a command-line argument parser used by the program main method.

Returns

Return type ArgumentParser

`examples.frontend_poll.`**`main`**(*args=['-b', 'latex', '-D', 'language=en', '-d', '_build/doctrees', '.', '_build/latex']*)

Parses arguments and passes them to *run()* method. Returns one when an error occurs.

Returns

Return type int

`examples.frontend_poll.`**`run`**(*client*)

Raises *UnexpectedMqttEventError*

Parameters **client** (ExampleMqttClient) –

4.1.5 Building Documentation

```
$ pip install sphinxcontrib-seqdiag
$ make html
$
```

4.1.6 Semantic Versioning

The *mqtt-codec* package is versioned according to Semantic Versioning 2.0.0 guidelines. A summary of SemVer is included here for your convenience:

Given a version number MAJOR.MINOR.PATCH, increment the:

1. MAJOR version when you make incompatible API changes,

2. MINOR version when you add functionality in a backwards-compatible manner, and
3. PATCH version when you make backwards-compatible bug fixes.

Additional labels for pre-release and build metadata are available as extensions to the MAJOR.MINOR.PATCH format.

—Semantic Versioning Summary, <https://semver.org/#summary>, retrieved 2018-10-01.

4.1.7 Requirements

- Python 2.7, with enum34
- Python 3.0, with enum34
- Python 3.1, with enum34
- Python 3.2, with enum34
- Python 3.3, with enum34
- Python 3.4
- Python 3.5
- Python 3.6
- Python 3.7

mqtt-codec>=1.0.2

4.1.8 "Weapons Grade"

The *haka-mqtt* project was a response to reliability problems and the absence of thorough documentation in existing python MQTT clients at the time. The authors were involved in projects that required thousands of clients to operate for years without service in soft realtime conditions over unreliable low-bandwith bandwidth data links. There was great desire to use off the shelf libraries but after two years of trying it was decided that a new approach, *haka-mqtt*, was needed.

The new *haka-mqtt* library provides at its core a reactor built to be thoroughly and deterministically tested. Its memory usage is stable and well documented. It is built to use non-blocking sockets in a select-type M:N thread model where M client tasks are mapped onto N threads. When suitable non-blocking APIs are not available (as for DNS lookups), those operations can be performed in separate thread-pools so that the client can continue its non-blocking operation.

Due to persistent questions it is necessary to state that the project should not be used in conjunction with enriched uranium, TNT, or any form of cruise missile.

haka

A primary purpose of software is to provide its authors the opportunity to disguise elaborate puns as real work. Eclipse's paho project provides MQTT client implementations and the verb "pāho" means to broadcast or announce (Maori Dictionary). What then is a "weapon's grade announcement"? The culturally ignorant authors contend that the haka dance performed by the Maori is truly a weapons grade announcement. The authors are also jealous and wish that they could participate.

4.2 API Reference

4.2.1 haka_mqtt.clock module

class haka_mqtt.clock.**SettableClock**

Bases: `object`

add_time(*duration*)

set_time(*t*)

time()

class haka_mqtt.clock.**SystemClock**

Bases: `object`

time()

4.2.2 haka_mqtt.cycle_iter module

class haka_mqtt.cycle_iter.**IntegralCycleIter**(*start*, *end*)

Bases: `object`

Parameters

- **start** (*int*) –
- **end** (*int*) –

next()

Returns the next iterator in the sequence.

Returns

Return type int

4.2.3 haka_mqtt.dns_async module

class haka_mqtt.dns_async.**AsyncFutureDnsResolver**(*thread_pool_size=1*)

Bases: `object`

An executor that spawns a small thread pool for performing DNS lookups. DNS lookup task sare submitted by using calling this object as a function and those tasks will be completed asynchronously by threads in a thread pool. The completed tasks are posted back to an internal queue and the `poll()` method gets the completed tasks from the queue and notifies their subscribers of completion. Tasks with done callback methods will be called by poll on the same threat that poll is called on.

```
>>> resolver = AsyncFutureDnsResolver()
>>>
>>> lookup_result = None
>>>
>>> def lookup_finished(future):
...   global lookup_result
...   lookup_result = future.result()
...
>>> future = resolver('localhost', 80)
>>> future.add_done_callback(lookup_finished)
>>>
```

(continues on next page)

(continued from previous page)

```
>>> while not future.done():
...   # Calling poll services callbacks and sets future to done.
...   resolver.poll()
...   sleep(0.1)
...
>>> assert future.done()
>>>
>>> # Only timeout=0 presently supported.
>>> assert not future.exception(timeout=0)
>>> assert not future.cancelled()
>>> # future.result(0) contains outcome of asynchronous call
>>> # to socket.getaddrinfo.  Note that at this time only timeout=0
>>> # is supported by this limited api.
```

```
                           |                      Worker Threads
                           |                      +-------------+
                           |                  +-->| Worker 0    |-->+
                           |                  |   +-------------+   |
                           |                  |                     |
                           |                  |   +-------------+   |
                           |                  +-->| Worker 1    |-->+
                           |                  |   +-------------+   |
                           |                  |                     |
                           |                  |   +-------------+   |
resolver('google.com', 80) |--> task queue----+-->| Worker ...  |-->+
                           |                  |   +-------------+   |
                           |                  |                     |
    AsyncFutureDnsResolver |                  |   +-------------+   |
                    Thread |                  +-->| Worker n    |-->+
                           |                      +-------------+   |
                           |                                        |
                           |                                        |
          resolver.poll()  |<-- completion queue<-------------------+
                           |
```

close()
: Closes resolver by completing all tasks in queue and joining with worker threads. New dns resolutions cannot be scheduled after this method begins executing (calling the resolver will result in an assertion failure).

closed()
: bool: True if the object has been closed; False otherwise.

poll()
: Calls done callbacks of any newly completed futures.

read_fd()
: int: fileno

4.2.4 haka_mqtt.dns_sync module

class haka_mqtt.dns_sync.**SynchronousFuture**(*result=None*, *exception=None*)

Bases: object

add_done_callback(*fn*)
: Attaches the callable fn to the future. fn will be called, with the future as its only argument, when the

future is cancelled or finishes running.

Added callables are called in the order that they were added and are always called in a thread belonging to the process that added them. If the callable raises an Exception subclass, it will be logged and ignored. If the callable raises a BaseException subclass, the behavior is undefined.

If the future has already completed or been cancelled, fn will be called immediately.

`cancel``()`

Always returns False since this future is finished the instant it is created.

Attempt to cancel the call. If the call is currently being executed and cannot be cancelled then the method will return False, otherwise the call will be cancelled and the method will return True.

Returns

Return type bool

`cancelled``()`

Always returns False since this future is finished the instant it is created.

Return True if the call was successfully cancelled.

Returns

Return type bool

`done``()`

Always returns true since this future is finished the instant it is created.

Return True if the call was successfully cancelled or finished running.

Returns

Return type bool

`exception``(`*timeout=None*`)`

Immediately returns the exception raised by the call or None if the call completed without raising.

Return the exception raised by the call. If the call hasn't yet completed then this method will wait up to timeout seconds. If the call hasn't completed in timeout seconds, then a concurrent.futures.TimeoutError will be raised. timeout can be an int or float. If timeout is not specified or None, there is no limit to the wait time.

If the future is cancelled before completing then CancelledError will be raised.

If the call completed without raising, None is returned.

`result``(`*timeout=None*`)`

Immediately returns the call resultor None if the call raised an exception.

Return the value returned by the call. If the call hasn't yet completed then this method will wait up to timeout seconds. If the call hasn't completed in timeout seconds, then a concurrent.futures.TimeoutError will be raised. timeout can be an int or float. If timeout is not specified or None, there is no limit to the wait time.

If the future is cancelled before completing then CancelledError will be raised.

If the call raised, this method will raise the same exception.

`class` `haka_mqtt.dns_sync.`**`SynchronousFutureDnsResolver`**

Bases: `object`

4.2.5 haka_mqtt.exception module

exception haka_mqtt.exception.**PacketIdReactorException**
Bases: *haka_mqtt.exception.ReactorException*

exception haka_mqtt.exception.**ReactorException**
Bases: exceptions.Exception

4.2.6 haka_mqtt.on_str module

class haka_mqtt.on_str.**HexOnStr**(*buf*)
Bases: object

class haka_mqtt.on_str.**ReprOnStr**(*o*)
Bases: object

4.2.7 haka_mqtt.reactor module

The reactor module provides an MQTT reactor class suitable for use with select and select-like interfaces like epoll. An adapter is available to make it conveniently usable in a poll environment (*haka_mqtt.frontends.poll*).

class haka_mqtt.reactor.**AddressReactorError**(*gaierror*)
Bases: *haka_mqtt.reactor.ReactorError*

Failed to lookup a valid address.

Parameters **gaierror** (*socket.gaierror*) –

gaierror
socket.gaierror – Addressing error.

class haka_mqtt.reactor.**ConnectReactorError**(*result*)
Bases: *haka_mqtt.reactor.ReactorError*

Error that occurs when the server sends a connack fail in response to an initial connect packet.

Parameters **result** (*ConnackResult*) – Asserted not to be *ConnackResult.accepted*.

result
ConnackResult – guaranteed that value is not *ConnackResult.accepted*.

class haka_mqtt.reactor.**DecodeReactorError**(*description*)
Bases: *haka_mqtt.reactor.ReactorError*

Server wrote a sequence of bytes that could not be interpreted as an MQTT packet.

class haka_mqtt.reactor.**MqttState**
Bases: enum.IntEnum

Inactive states are those where there are no active deadlines, the socket is closed and there is no active I/O. Active states are those where any of these characteristics is not met.

Active States:

- *MqttState.connack*
- *MqttState.connected*
- *MqttState.mute*

Inactive States:

- *MqttState.stopped*

connack = 0

connected = 1

mute = 2

stopped = 3

class haka_mqtt.reactor.**MutePeerReactorError**

Bases: *haka_mqtt.reactor.ReactorError*

Error that occurs when the server closes its write stream unexpectedly.

class haka_mqtt.reactor.**ProtocolReactorError**(*description*)

Bases: *haka_mqtt.reactor.ReactorError*

Server send an inappropriate MQTT packet to the client.

class haka_mqtt.reactor.**Reactor**(*properties*, *log='haka'*)

Bases: `object`

Parameters

- **properties** (ReactorProperties) –
- **log** (*str or logging.Logger or None*) – If *str* then the result of logging.getLogger(log) is used as a logger; otherwise assumes that is a *logging.Logger*-like object and asserts that it has *debug*, *info*, *warning*, *error*, and *critical* methods. If *log* is *None* then logging is disabled.

clean_session

bool – Clean session flag is true/false.

client_id

str – Client id.

error

ReactorError or None – When *self.state* is *ReactorState.error* returns a subclass of *ReactorError* otherwise returns *None*.

in_flight_packets()

is_active()

`True` when reactor is active; `False` otherwise.

An "active" reactor implies that there are outstanding scheduler deadlines active, possibly open sockets, or possibly outstanding DNS lookup futures. This method would return `True` for this case.

An inactive reactor guarantees that there are no oustanding scheduler deadlines, DNS lookups, or open sockets. An inactive reactor will never change state unless a method like *start()* is called to start the reactor.

New in version 0.3.5.

Returns

Return type bool

keepalive_period

int – If this period elapses without the client sending a control packet to the server then it will generate a pingreq packet and send it to the server. Will return zero if pingreq requests are not generated.

mqtt_state

MqttState – Current state of mqtt protocol handshake.

on_connack(*reactor*, *connack*)
Called immediately upon receiving a *MqttConnack* packet from the remote. The *reactor.state* will be *ReactorState.started* or *ReactorState.stopping* if the reactor is shutting down.

Parameters

- **reactor** (`Reactor`) –
- **connack** (`mqtt_codec.packet.MqttConnack`) –

on_connect_fail(*reactor*)

Parameters **reactor** (`Reactor`) –

on_disconnect(*reactor*)

Parameters **reactor** (`Reactor`) –

on_puback(*reactor*, *puback*)
Called immediately upon receiving a *MqttPuback* packet from the remote. This method is part of the QoS=1 message send path.

Parameters

- **reactor** (`Reactor`) –
- **puback** (`mqtt_codec.packet.MqttPuback`) –

on_pubcomp(*reactor*, *pubcomp*)
Called immediately upon receiving a *MqttPubcomp* packet from the remote. This is part of the QoS=2 message send path.

Parameters

- **reactor** (`Reactor`) –
- **pubcomp** (`mqtt_codec.packet.MqttPubcomp`) –

on_publish(*reactor*, *publish*)
Called immediately upon receiving a *MqttSuback* packet from the remote. This is part of the QoS=0, 1, and 2 message receive paths.

Parameters

- **reactor** (`Reactor`) –
- **publish** (`mqtt_codec.packet.MqttPublish`) –

on_pubrec(*reactor*, *pubrec*)
Called immediately upon receiving a *MqttPubrec* packet from the remote. This is part of the QoS=2 message send path.

Parameters

- **reactor** (`Reactor`) –
- **pubrec** (`mqtt_codec.packet.MqttPubrec`) –

on_pubrel(*reactor*, *pubrel*)
Called immediately upon receiving a *MqttPubrel* packet from the remote. This is part of the QoS=2 message receive path.

Parameters

- **reactor** (`Reactor`) –
- **pubrel** (`mqtt_codec.packet.MqttPubrel`) –

on_suback(*reactor*, *suback*)
Called immediately upon receiving a *MqttSuback* packet from the remote.

Parameters

- **reactor** (`Reactor`) –
- **suback** (`mqtt_codec.packet.MqttSuback`) –

on_unsuback(*reactor*, *unsuback*)
Called immediately upon receiving a *MqttUnsuback* packet from the remote.

Parameters

- **reactor** (`Reactor`) –
- **unsuback** (`mqtt_codec.packet.MqttUnsuback`) –

preflight_packets()

publish(*topic*, *payload*, *qos*, *retain=False*)
Places a publish packet on the preflight queue. Messages in the preflight queue are fair-queued and launched to the server. The reactor certainly will try to place as many messages in-flight as it is able to. If you want to limit the number of messages in-flight then a queue should be maintained outside of the core reactor.

QoS 0 messages are placed in the pre-flight buffer and are eligable for delivery as fast as the socket allows. If the reactor encounters an error or stops and is subsequently started then any QoS=0 messages in the preflight queue are discarded. QoS 0 messages are considered delivered as soon as one of their bytes is placed in the socket write buffer regardless of whether the network successfully delivers them to their destination.

QoS 1 messages are placed in the pre-flight buffer and are eligable for delivery as fast as the socket allows. They are placed in the in-flight queue as soon as the first byte of the packet is placed in the socket write buffer. If the reactor encounters an error or stops and is subsequently started then any QoS=1 messages in the preflight queue maintain their positions. Any messages in the in-flight queue are placed in the front of the preflight queue as `publish` packets with their dupe flags set to `True`.

QoS 2 messages are placed in the pre-flight buffer and are eligable for delivery as fast as the socket allows. They are placed in the in-flight queue as soon as the first byte of the packet is placed in the socket write buffer. If the reactor encounters an error or stops and is subsequently started then any QoS=2 messages in the preflight queue maintain their positions. Any messages in the in-flight queue awaiting `pubrec` acknowledgements are placed in the front of the preflight queue as publish packets with their dupe flags set to `True`. Any messages in the in-flight queue awaiting `pubcomp` acknowledgements are placed in the front of the preflight queue as `pubrel` packets.

Parameters

- **topic** (*str*) –
- **payload** (*bytes*) –
- **qos** (*int*) – 0 <= qos <= 2
- **retain** (*bool*) –

Raises *haka_mqtt.exception.PacketIdReactorException* – Raised when there are no free packet ids to create a *MqttPublish* packet with.

Returns A publish ticket. The returned object will satisfy *ticket.status is MqttPublishStatus.preflight*.

Return type MqttPublishTicket

read()

Calls recv on underlying socket exactly once and returns the number of bytes read. If the underlying socket does not return any bytes due to an error or exception then zero is returned and the reactor state is set to error.

This method may be called at any time in any state and if *self* is not prepared for a read at that point then no action will be taken.

The *socket.settimeout* can be used to perform a blocking read with a timeout on the underlying socket.

Returns number of bytes read from socket.

Return type int

recv_idle_abort_period

int – Connection will be closed if bytes have not been received from remote in this many seconds. Typically this is 1.5x the self.keepalive_period.

recv_idle_ping_period

int – 0 <= `self.recv_idle_ping_period`; sends a `mqtt_codec.packet.MqttPingreq` packet to the server after this many seconds without receiving and bytes on the socket. If zero then ping messages are not sent when receive stream is idle.

send_packet_ids()

Returns A set of active send-path packet ids.

Return type set[int]

sock_state

SocketState – Current state of the socket connection.

start()

Attempts to connect with remote if in one of the inactive states *`ReactorState.init`*, *`ReactorState.stopped`*, *`ReactorState.error`*. The method has no effect if already in an active state.

state

ReactorState – Current reactor state.

stop()

subscribe(*topics*)

Places a `subscribe` packet on the preflight queue. Messages in the preflight queue will be placed in-flight as soon as the socket allows. Multiple messages may be placed in-flight at the same time.

If the reactor encounters an error or stops then unacknowledged `subscribe` packets will be dropped whether they are in the preflight or the in-flight queues.

Parameters **topics** (*iterable of MqttTopic*) –

Raises *`haka_mqtt.exception.PacketIdReactorException`* – Raised when there are no free packet ids to create a *MqttSubscribe* packet with.

Returns

Return type MqttSubscribeTicket

terminate()

When in an active state immediately shuts down any socket reading and writing, closes the socket, cancels all outstanding scheduler deadlines, puts the reactor into state ReactorState.stopped, then calls self.on_connect_fail (if in a connect/connack state) or alternatively self.on_disconnect if in some other active state. When reactor is not in an inactive state this method has no effect.

unsubscribe(*topics*)

Places an `unsubscribe` packet on the preflight queue. Messages in the preflight queue will be placed in-flight as soon as the socket allows. Multiple messages may be placed in-flight at the same time.

If the reactor encounters an error or stops then unacknowledged `unsubscribe` packets will be dropped whether they are in the preflight or the in-flight queues.

Parameters **topics** (*iterable of str*) –

Raises *haka_mqtt.exception.PacketIdReactorException* – Raised when there are no free packet ids to create a *MqttUnsubscribe* packet with.

Returns

Return type MqttUnsubscribeTicket

want_read()

True if the reactor is ready to process incoming socket data; False otherwise.

Returns

Return type bool

want_write()

True if the reactor is ready write data to the socket; False otherwise.

Returns

Return type bool

will

`mqtt_codec.packet.MqttWill` or `None` – Last will and testament.

write()

If there is any data queued to be written to the underlying socket then a single call to socket send will be made to try and flush it to the socket write buffer.

This method may be called at any time in any state and if *self* is not prepared for a write at that point then no action will be taken.

The *socket.settimeout* can be used to perform a blocking write with a timeout on the underlying socket.

class haka_mqtt.reactor.**ReactorError**

Bases: `object`

class haka_mqtt.reactor.**ReactorProperties**

Bases: `object`

socket_factory

haka_mqtt.socket_factory.SocketFactory

name_resolver

callable – DNS resolver.

scheduler

TODO

selector

Selector

client_id

str

endpoint

tuple – 2-tuple of (host: *str*, port: *int*). The *port* value is constrainted such that 0 <= *port* <= 2**16-1.

keepalive_period
int – 0 <= keepalive_period <= 2*16-1; zero disables keepalive. Sends a `mqtt_codec.packet.MqttPingreq` packet to the server after this many seconds without sending and data over the socket. The server will disconnect the client as if there has been a network error after 1.5x‘‘self.keepalive_period‘‘ seconds without receiving any bytes [MQTT-3.1.2-24].

recv_idle_ping_period
int – 0 < recv_idle_ping_period; sends a `mqtt_codec.packet.MqttPingreq` packet to the server after this many seconds without receiving and bytes on the socket.

recv_idle_abort_period
int – 0 < recv_idle_abort_period; aborts connection after this time without receiving any bytes from remote (typically set to 1.5x `self.recv_idle_ping_period`).

clean_session
bool – With clean session set to True reactor will clear all message buffers on disconnect without regard to QoS; otherwise unacknowledged messages will be retransmitted after a re-connect.

address_family
int – Address family; one of the socket.AF_* constants (eg. `socket.AF_UNSPEC` for any family, `socket.AF_INET` for IP4 `socket.AF_INET6` for IP6). Set to `socket.AF_UNSPEC` by default.

username
str optional

password
str optional

class `haka_mqtt.reactor.`**ReactorState**
Bases: `enum.IntEnum`

Inactive states are those where there are no active deadlines, the socket is closed and there is no active I/O. Active states are those where any of these characteristics is not met.

Active States:

- *`ReactorState.init`*
- *`ReactorState.stopped`*
- *`ReactorState.error`*

Inactive States:

- `ReactorState.connecting`
- `ReactorState.handshake`
- `ReactorState.connack`
- `ReactorState.connected`

error = 5

init = 0

started = 2

starting = 1

stopped = 4

stopping = 3

class haka_mqtt.reactor.**RecvTimeoutReactorError**
Bases: *haka_mqtt.reactor.ReactorError*

Server fails to respond in a timely fashion.

class haka_mqtt.reactor.**SocketReactorError**(*errno_val*)
Bases: *haka_mqtt.reactor.ReactorError*

A socket.error exception was raised by the socket subsystem and it the error code was *self.errno*. If this errno is in the *errno.errorcode* lookup table then repr will show the description.

Parameters **errno_val** (*int*) –

errno
int – value in *errno.errorcode*.

class haka_mqtt.reactor.**SocketState**
Bases: enum.IntEnum

Inactive states are those where there are no active deadlines, the socket is closed and there is no active I/O. Active states are those where any of these characteristics is not met.

Active States:

- *SocketState.name_resolution*
- *SocketState.connecting*
- *SocketState.handshake*
- *SocketState.connected*
- *SocketState.deaf*
- *SocketState.mute*

Inactive States:

- *SocketState.stopped*

connected = 5

connecting = 2

deaf = 6

handshake = 3

mute = 7

name_resolution = 1

stopped = 8

class haka_mqtt.reactor.**SslReactorError**(*ssl_error*)
Bases: *haka_mqtt.reactor.ReactorError*

A socket error-code in *errno.errorcode*.

Parameters **ssl_error** (*ssl.SSLError*) –

error
ssl.SSLError – error value.

4.2.8 haka_mqtt.frontends.poll module

class haka_mqtt.frontends.poll.**BlockingMqttClient**(*properties*, *log='haka'*)
Bases: *haka_mqtt.reactor.Reactor*

A client that employs socket.settimeout to use blocking operations on sockets. Although in general the socket timeout is respected by the operating system, this method uses a synchronous DNS lookup and this lookup does not have any timeout.

Parameters **properties** (`MqttPollClientProperties`) –

poll(*period=0.0*)

class haka_mqtt.frontends.poll.**MqttPollClient**(*properties*, *log='haka'*)
Bases: *haka_mqtt.reactor.Reactor*

Parameters **properties** (`MqttPollClientProperties`) –

poll(*period=0.0*)

class haka_mqtt.frontends.poll.**MqttPollClientProperties**
Bases: `object`

client_id
str or None – The MQTT client id to pass to the MQTT server. If *None* then a client-id will be generated with A client id will be randomly generated based on *generate_client_id()*.

address_family
int – Address family; one of the socket.AF_* constants (eg. `socket.AF_UNSPEC` for any family, `socket.AF_INET` for IP4 `socket.AF_INET6` for IP6). By default this will be `socket.AF_UNSPEC`.

host
str – IP address or host name.

port
int – Integer such that 0 <= port <= 2**16-1.

keepalive_period
str – 0 <= keepalive_period <= 2*16-1; zero disables keepalive. Sends a `MqttPingreq` packet to the server after this many seconds without sending and data over the socket. The server will disconnect the client as if there has been a network error after 1.5x‘‘self.keepalive_period‘‘ seconds without receiving any bytes [MQTT-3.1.2-24].

recv_idle_ping_period
int – 0 < recv_idle_ping_period; sends a `MqttPingreq` packet to the server after this many seconds without receiving and bytes on the socket.

recv_idle_abort_period
int – 0 < recv_idle_abort_period; aborts connection after this time without receiving any bytes from remote (typically set to 1.5x `self.recv_idle_ping_period`).

ssl
bool or SSLContext – When *True* connects to server using a default SSL socket context created with `ssl.create_default_context()`.

If `ssl` has a callable `wrap_socket` method then it is assumed that `ssl` is a SSLContext to be used for securing sockets.

haka_mqtt.frontends.poll.**generate_client_id**()
Generates a client id based on current time, hostname, and process-id.

Returns

Return type str

4.2.9 haka_mqtt.scheduler module

class haka_mqtt.scheduler.**ClockScheduler**(*clock*)

Bases: *haka_mqtt.scheduler.Scheduler*

instant()

Current clock instant.

Returns Current clock scheduler.

Return type int

poll()

Calls all callbacks awaiting execution to this point.

class haka_mqtt.scheduler.**Deadline**(*deadline_entry*)

Bases: object

cancel()

Stops a scheduled callback from being made; has no effect if cancel is called after the callback has already been made.

expired()

bool: True if callback has already been called; False otherwise.

class haka_mqtt.scheduler.**DurationScheduler**

Bases: *haka_mqtt.scheduler.Scheduler*

instant()

Returns the current tick.

Returns

Return type int

poll(*duration*)

Adds *duration* to *self.instant()* and calls all scheduled callbacks.

Parameters **duration** (*int*) –

class haka_mqtt.scheduler.**Scheduler**

Bases: object

add(*duration*, *cb*)

Adds *duration* to *self.instant()* and calls all scheduled callbacks.

Parameters

- **duration** (*int*) – Number of ticks passed.
- **cb** (*callable()*) – No calling with so

Returns

Return type *Deadline*

instant()

Returns the current tick.

Returns

Return type int

remaining()
Duration remaining to next scheduled callback.

Returns

Return type int or None

4.2.10 haka_mqtt.selector module

class haka_mqtt.selector.**Selector**
Bases: object

add_read(*fd*, *reactor*)

Parameters

- **fd** (*file descriptor*) – File-like object.
- **reactor** (haka_mqtt.reactor.Reactor) –

add_write(*f*, *reactor*)

Parameters

- **fd** (*file descriptor*) – File-like object.
- **reactor** (haka_mqtt.reactor.Reactor) –

del_read(*fd*, *reactor*)

Parameters

- **fd** (*file descriptor*) – File-like object.
- **reactor** (haka_mqtt.reactor.Reactor) –

del_write(*fd*, *reactor*)

Parameters

- **fd** (*file descriptor*) – File-like object.
- **reactor** (haka_mqtt.reactor.Reactor) –

4.2.11 haka_mqtt.socket_factory module

class haka_mqtt.socket_factory.**BlockingSocketFactory**
Bases: object

class haka_mqtt.socket_factory.**BlockingSslSocketFactory**(*context*)
Bases: object

class haka_mqtt.socket_factory.**SocketFactory**
Bases: object

class haka_mqtt.socket_factory.**SslSocketFactory**(*context*)
Bases: object

4.3 Change Log

4.3.1 0.3.5 (2019-02-12)

New

#35: Added is_active method to core reactor.

Determining whether a reactor had oustanding socket I/O, scheduler deadlines or futures was:

if reactor.state in ACTIVE_STATES: pass # Do stuff

but this is less obvious than:

if reactor.is_active(): pass # Do stuff

The change should be helpful particularly to new users of the library.

https://github.com/kcallin/haka-mqtt/issues/35

Fix

#27: MQTT 3.1.1, MQTT-2.3.1-1, Packet IDs must be non-zero.

Packet ID generation starts at zero and this is a violation of MQTT 3.1.1 MQTT-2.3.1-1. Packet IDs are now generated beginning with one instead of zero.

https://github.com/kcallin/haka-mqtt/issues/27

#30: BlockingMqttClient must use BlockingSslSocketFactory for ssl.

When uses incorrect socket factory when properties.ssl is an SSLContext BlockingMqttClient incorrectly uses a SslSocketFactory instead of a BlockingSslSocketFactory.

https://github.com/kcallin/haka-mqtt/issues/30

#31: Log chosen DNS entry at INFO; others at DEBUG.

On DNS lookup one entry is chosen as the endpoint and other entries are logged for information purposes. This leads to noisy logs. Chosen entry now logged at INFO and others are logged at DEBUG level.

https://github.com/kcallin/haka-mqtt/issues/31

#32: DNS lookup of IP6 addresses logs address in square brackets.

When logging IP6 addresses from DNS lookups the addresses should be enclosed by square brackets "[]" to distinguish them from port numbers.

Before: 2001:db8:85a3:8d3:1319:8a2e:370:7348:443 After: [2001:db8:85a3:8d3:1319:8a2e:370:7348]:443

https://github.com/kcallin/haka-mqtt/issues/32

#34: MqttPingreq not always scheduled correctly.

Client is not permitted to have more than one unacknowledged pingreq at any given time. When a keepalive pingreq comes due and the acknowledging pingresp is delayed (by say more than keepalive period) the reactor needs to be able to immediately launch a pingreq upon receipt of the pingresp.

https://github.com/kcallin/haka-mqtt/issues/34

4.3.2 0.3.4 (2019-01-31)

New

- BlockingMqttClient and MqttPollClient now support a SSLContext parameter.

Fix

#28: Assertion failure in __recv_idle_abort_timeout handler.

If an recv_idle_abort_timeout occurs while the keepalive_timeout is active then an assertion fails resulting in a library crash. The assertion was incorrectly placed and has been removed. Bug has existed since 0.3.0.

https://github.com/kcallin/haka-mqtt/issues/28

4.3.3 0.3.3 (2019-01-29)

Fix

#22: On p3k DNS async resolver does not pass bytes to write.

On Python 3 the DNS async resolver passes str instead of bytes to an os.write call. This results in a TypeError and crash.

https://github.com/kcallin/haka-mqtt/issues/22

#24: On p3k HexOnStr __str__ method fails to return a str.

On Python 3 str(HexOnStr) fails to return str and this results in a TypeError.

https://github.com/kcallin/haka-mqtt/issues/24

#25: socket.timeout has None errno; fails assertion.

When using reactor in blocking mode with timeouts, socket.timeout exceptions can be raised. This is a subclass of socket.error and caught as such. The reactor asserts that socket.error has meaningful errno so this crashes.

https://github.com/kcallin/haka-mqtt/issues/25

#26: Poll frontends guarantee at least one read/write per poll call.

When MqttPollClient/MqttBlockingClient were called, they should guarantee at least one read/write call per poll. This way when the poll period is set to zero the read/write call will still be called (previously it might not be as clocks would timeout before the read/write call was made.

https://github.com/kcallin/haka-mqtt/issues/26

4.3.4 0.3.2 (2019-01-13)

Fix

#21: haka_mqtt/frontends/poll.py not included in build

The polling frontend was mistakenly left out of the pypi package.

https://github.com/kcallin/haka-mqtt/issues/21

4.3.5 0.3.1 (2018-12-30)

New

#20: Remove ordering restrictions on QoS=2 send path.

https://github.com/kcallin/haka-mqtt/issues/20

Fix

#17: Connect fail after DNS lookup fails to enter error state.

After a DNS lookup succeeds but the subsequent socket connect fails core reactor may not enter the error state.

https://github.com/kcallin/haka-mqtt/issues/17

#18: Haka crash when SSL raises socket.error with zero errno.

Some SSL subsystems can raise socket.error exceptions with zero errno values. This fails one of haka's assertions. The assertion has been removed and the SocketReactorError class description has been changed.

https://github.com/kcallin/haka-mqtt/issues/18

#19: Full socket buffer can trigger message retransmissions.

When the socket buffer is full and a call to send returns with zero bytes then a second copy of the message may be queued in the reactor write buffer. The end result is that the message can be placed in flight more than once.

https://github.com/kcallin/haka-mqtt/issues/19

4.3.6 0.3.0 (2018-12-17)

It is recommended to update to update to 0.3.0 immediately to avoid a crash as a result of #16.

New

#15: Support disabling Reactor.recv_idle_ping_period. https://github.com/kcallin/haka-mqtt/issues/15

Fix

#16: Keepalive scheduled while pingreq already active.

If a write operation is triggered with a pingreq in-the-air then the reactor incorrectly schedules a new pingreq. There is no danger of a new pingreq being launched but if a recv_idle_abort_timeout occurs while in this condition an assertion fails.

This is a crashing bug.

https://github.com/kcallin/haka-mqtt/issues/16

4.3.7 0.2.0 (2018-11-29)

New

#9: Run without keepalive. https://github.com/kcallin/haka-mqtt/issues/9

Fix

#13: trigger keepalive on send instead of recv. https://github.com/kcallin/haka-mqtt/issues/13

4.3.8 0.1.0 (2018-10-25)

- Initial release.

4.4 Developer Guide

The developer's guide is for a person who wants to change and contribute changes to *haka-mqtt*. It builds on information in *User Guide*.

4.4.1 Uncontrolled Builds

Uncontrolled source builds are created in the standard python fashion:

```
$ python setup.py sdist
running sdist
running egg_info
writing requirements to haka_mqtt.egg-info/requires.txt
writing haka_mqtt.egg-info/PKG-INFO
writing top-level names to haka_mqtt.egg-info/top_level.txt
writing dependency_links to haka_mqtt.egg-info/dependency_links.txt
reading manifest file 'haka_mqtt.egg-info/SOURCES.txt'
writing manifest file 'haka_mqtt.egg-info/SOURCES.txt'
running check
creating haka-mqtt-0.1.0-uncontrolled-20180907
creating haka-mqtt-0.1.0-uncontrolled-20180907/haka_mqtt
creating haka-mqtt-0.1.0-uncontrolled-20180907/haka_mqtt.egg-info
[... removed for brevity ...]
copying tests/test_reactor.py -> haka-mqtt-0.1.0-uncontrolled-20180907/tests
copying tests/test_scheduler.py -> haka-mqtt-0.1.0-uncontrolled-20180907/tests
Writing haka-mqtt-0.1.0-uncontrolled-20180907/setup.cfg
creating dist
Creating tar archive
removing 'haka-mqtt-0.1.0-uncontrolled-20180907' (and everything under it)
$ ls dist
haka-mqtt-0.1.0-uncontrolled-20180907.tar.gz
$
```

The output artifact has the word "uncontrolled" along with a build date so that users will know the artifact is not a release or from a continuous integration build server.

4.4.2 Tests

The *haka-mqtt* library comes with an extensive battery of tests. The tests are built to be as deterministic as possible - to the point that the loggers are not connected to the system clock so that the time of a given log message in the tests will be identical from one test run to the next.

The built-in automated tests can be run from the command-line in the conventional python manner:

```
$ python setup.py test
$
```

What is less conventional is that logging to standard output can be enabled by setting the *LOGGING* environment variable.

```
$ LOGGING=true python setup.py test
$
```

4.4.3 Coverage

Test coverage is monitored using coverage.py version 4.5 or higher. Normally this can be installed through your operating system's package manager (like rpm or apt-get) or by using *pip*. A coverage configuration file is included at *.coveragerc* and the tool can be run in this fashion:

```
$ coverage run setup.py test
running test
Searching for mock
Best match: mock 2.0.0
Processing mock-2.0.0-py2.7.egg

Using /home/kcallin/src/haka-mqtt/.eggs/mock-2.0.0-py2.7.egg
Searching for pbr>=0.11
Best match: pbr 4.2.0
Processing pbr-4.2.0-py2.7.egg

Using /home/kcallin/src/haka-mqtt/.eggs/pbr-4.2.0-py2.7.egg
running egg_info
writing requirements to haka_mqtt.egg-info/requires.txt
writing haka_mqtt.egg-info/PKG-INFO
writing top-level names to haka_mqtt.egg-info/top_level.txt
writing dependency_links to haka_mqtt.egg-info/dependency_links.txt
reading manifest file 'haka_mqtt.egg-info/SOURCES.txt'
writing manifest file 'haka_mqtt.egg-info/SOURCES.txt'
running build_ext
test_connack (tests.test_reactor_keepalive.TestKeepalive) ... ok
test_connected (tests.test_reactor_keepalive.TestKeepalive) ... ok
test_connected_keepalive_with_recv_qos0 (tests.test_reactor_keepalive.TestKeepalive) .
↪.. ok
test_connected_unsolicited_pingresp (tests.test_reactor_keepalive.TestKeepalive) ...␣
↪ok
[... removed for brevity...]
test_repeat (tests.test_cycle_iter.TestIterCycles) ... ok
test_scheduler (tests.test_scheduler.TestScheduler) ... ok
test_scheduler_0 (tests.test_scheduler.TestScheduler) ... ok

----------------------------------------------------------------------
Ran 95 tests in 0.376s
```

(continues on next page)

(continued from previous page)

```
OK
$ coverage report
Name                            Stmts   Miss Branch BrPart  Cover
-------------------------------------------------------------
haka_mqtt/__init__.py               0      0      0      0   100%
haka_mqtt/clock.py                 13      1      0      0    92%
haka_mqtt/cycle_iter.py            16      0      2      0   100%
haka_mqtt/exception.py              4      0      0      0   100%
haka_mqtt/mqtt_request.py          89      2      6      2    96%
haka_mqtt/null_log.py              15      7      0      0    53%
haka_mqtt/on_str.py                17      3      4      1    71%
haka_mqtt/packet_ids.py            22      2      4      2    85%
haka_mqtt/reactor.py              945     83    316     29    90%
haka_mqtt/scheduler.py             75     11     12      1    84%
haka_mqtt/selector.py              11      0      0      0   100%
-------------------------------------------------------------
TOTAL                            1207    109    344     35    89%
```

4.4.4 Docstrings

Python source code is documented according to the the numpy documentation standard at https://numpydoc.readthedocs.io/en/latest/format.html.

4.4.5 Building Documentation

The documentation for mqtt-codec is created with Sphinx and is build the fashion usual to that framework:

```
$ pip install sphinxcontrib-plantuml enum34 mqtt-codec
$ cd doc
$ make html
$
```

4.4.6 Requirements

The project will eventually track requirements using a project like Pipfile.

4.5 Distributing haka-mqtt

The release procedure was created using information from these core sources:

- PEP 503 - Simple Repository API
- Python Packaging User Guide
- Twine

4.5.1 Documentation

Verify that version and release numbers in `doc/source/conf.py` match `setup.py`.

```
$ grep -e version -e release doc/source/conf.py
# The short X.Y version
version = u'1.0.0'
# The full version, including alpha/beta/rc tags
release = u'1.0.0'
$
```

Make sure copyright dates in `doc/source/conf.py` are correct:

```
$ grep -i copyright doc/source/conf.py
copyright = u'2018 - 2019, Keegan Callin'
$
```

Verify requirements document at `doc/source/requirements.rst` matches `setup.py` (and not the other way around).

4.5.2 Test Release

Clean build directory and ensure there are no old build artifacts.

```
$ rm -rf dist build haka_mqtt.egg-info htmlcov
$ ls dist
$
```

It's a common problem to accidentally forget to commit important changes. To combat this the `pyvertest.py` procedure clones the haka repository, passes it to a docker container, and runs a test battery in a set of environments.

```
$ ./pyvertest.py
[... removed for brevity ...]
pip install python:3.7-alpine3.8
docker run --rm -v /home/kcallin/src/haka-mqtt:/haka-mqtt python:3.7-alpine3.8 pip␣
↪install /haka-mqtt
Processing /haka-mqtt
Building wheels for collected packages: haka-mqtt, mqtt-codec
  Running setup.py bdist_wheel for haka-mqtt: started
  Running setup.py bdist_wheel for haka-mqtt: finished with status 'done'
  Stored in directory: /root/.cache/pip/wheels/c5/b3/fa/
↪e30017929f15cb43137c499453ff45f3754db112f34a52cb9d
  Running setup.py bdist_wheel for mqtt-codec: started
  Running setup.py bdist_wheel for mqtt-codec: finished with status 'done'
  Stored in directory: /root/.cache/pip/wheels/b7/6b/0f/
↪5fb8026a75541fb9fcdec2f3fc33b75aad929b48e85eca68a9
Successfully built haka-mqtt mqtt-codec
Installing collected packages: mqtt-codec, haka-mqtt
Successfully installed haka-mqtt-0.3.0-uncontrolled-20181217 mqtt-codec-1.0.1
Return code 0
Removing container id␣
↪b9d481a9f49b966fa6708e1ef9fda16d0142b35a7613fc794a43105b0eb6eb2b.
Removing temp directory /tmp/tmput2xuulf.
> 10/10 okay.
```

Ensure that CHANGELOG.rst has release version and release date correct as well as release content listed.

```
$ vi CHANGELOG.rst
$ git commit -S CHANGELOG.rst
```

Create test release artifacts.

```
$ python setup.py egg_info -D -b 'a' sdist
running sdist
running egg_info
writing requirements to haka_mqtt.egg-info/requires.txt
writing haka_mqtt.egg-info/PKG-INFO
writing top-level names to haka_mqtt.egg-info/top_level.txt
writing dependency_links to haka_mqtt.egg-info/dependency_links.txt
reading manifest file 'haka_mqtt.egg-info/SOURCES.txt'
writing manifest file 'haka_mqtt.egg-info/SOURCES.txt'
running check
creating haka-mqtt-0.1.2
creating haka-mqtt-0.1.2/haka_mqtt
[... removed for brevity ...]
copying tests/test_reactor.py -> haka-mqtt-0.1.2/tests
copying tests/test_scheduler.py -> haka-mqtt-0.1.2/tests
Writing haka-mqtt-0.1.2/setup.cfg
Creating tar archive
removing 'haka-mqtt-0.1.2' (and everything under it)
$ ls dist
haka-mqtt-0.1.2.tar.gz
$
```

GPG signatures are created for test release artifacts.

```
$ gpg2 --detach-sign -a dist/*

You need a passphrase to unlock the secret key for
user: "Keegan Callin <kc@kcallin.net>"
4096-bit RSA key, ID DD53792F, created 2017-01-01 (main key ID 14BC2EFF)

gpg: gpg-agent is not available in this session
$ ls dist
haka-mqtt-0.1.2.tar.gz  haka-mqtt-0.1.2.tar.gz.asc
$ gpg2 --verify dist/*.asc
gpg: assuming signed data in `dist/haka-mqtt-0.1.2.tar.gz'
gpg: Signature made Sat 01 Sep 2018 11:00:31 AM MDT using RSA key ID DD53792F
gpg: Good signature from "Keegan Callin <kc@kcallin.net>" [ultimate]
Primary key fingerprint: BD51 01F1 9699 A719 E563  6D85 4A4A 7B98 14BC 2EFF
     Subkey fingerprint: BE56 D781 0163 488F C7AE  62AC 3914 0AE2 DD53 792F
$
```

Ensure that twine version 1.12.0 or higher is installed:

```
$ twine --version
twine version 1.12.0 (pkginfo: 1.4.2, requests: 2.20.1, setuptools: 40.6.2,
requests-toolbelt: 0.8.0, tqdm: 4.28.1)
```

Verify that distribution passes twine checks:

```
$ twine check dist/*
Checking distribution dist/haka-mqtt-1.0.0.tar.gz: Passed
```

Release artifacts are uploaded to **TEST** PyPI.

```
$ twine upload --repository-url https://test.pypi.org/legacy/ dist/*
Uploading distributions to https://test.pypi.org/legacy/
Enter your username: kc
```

(continues on next page)

(continued from previous page)

```
Enter your password:
Uploading haka-mqtt-0.1.2.tar.gz
$
```

The resulting TestPyPI entry should be inspected for correctness. "The database for TestPyPI may be periodically pruned, so it is not unusual for user accounts to be deleted[1]". Packages on **TEST** PyPI and **real** PyPI cannot be removed upon distributor demand. On **TEST** PyPI packages may be removed on prune, on **real** PyPI they will remain forever. A checklist to help verify the PyPI release page follows:

- Version Number is Correct
- Documentation Link is Correct
- ReST README.rst is rendered correctly on the front page.

After the checklist is complete then it is time to upload to **real** PyPI and verify that the release is complete. There is no undoing this operation. Think Carefully.

PEP 508 – Dependency specification for Python Software Packages

PEP-314 – Metadata for Python Software Packages v1.1

4.5.3 Official Release

Create, sign, and push release tag:

```
$ git tag -s v0.1.0
$ git push origin v0.1.0
```

Remove test artifacts:

```
$ rm -rf dist build haka_mqtt.egg-info htmlcov
$ ls dist
$
```

Create official release artifacts.

```
$ python setup.py egg_info -D -b '' sdist
running sdist
running egg_info
writing requirements to haka_mqtt.egg-info/requires.txt
writing haka_mqtt.egg-info/PKG-INFO
writing top-level names to haka_mqtt.egg-info/top_level.txt
writing dependency_links to haka_mqtt.egg-info/dependency_links.txt
reading manifest file 'haka_mqtt.egg-info/SOURCES.txt'
writing manifest file 'haka_mqtt.egg-info/SOURCES.txt'
running check
creating haka-mqtt-0.1.2
creating haka-mqtt-0.1.2/haka_mqtt
[... removed for brevity ...]
copying tests/test_reactor.py -> haka-mqtt-0.1.2/tests
copying tests/test_scheduler.py -> haka-mqtt-0.1.2/tests
Writing haka-mqtt-0.1.2/setup.cfg
Creating tar archive
removing 'haka-mqtt-0.1.2' (and everything under it)
```

(continues on next page)

[1] Test PyPI, Registering Your Account, retrieved 2018-09-07.

(continued from previous page)

```
$ ls dist
haka-mqtt-0.1.2.tar.gz
$
```

GPG sign official release artifact:

```
$ gpg2 --detach-sign -a dist/*

You need a passphrase to unlock the secret key for
user: "Keegan Callin <kc@kcallin.net>"
4096-bit RSA key, ID DD53792F, created 2017-01-01 (main key ID 14BC2EFF)

gpg: gpg-agent is not available in this session
$ ls dist
haka-mqtt-0.1.2.tar.gz  haka-mqtt-0.1.2.tar.gz.asc
$ gpg2 --verify dist/*.asc
gpg: assuming signed data in `dist/haka-mqtt-0.1.2.tar.gz'
gpg: Signature made Sat 01 Sep 2018 11:00:31 AM MDT using RSA key ID DD53792F
gpg: Good signature from "Keegan Callin <kc@kcallin.net>" [ultimate]
Primary key fingerprint: BD51 01F1 9699 A719 E563  6D85 4A4A 7B98 14BC 2EFF
     Subkey fingerprint: BE56 D781 0163 488F C7AE  62AC 3914 0AE2 DD53 792F
$
```

The access credentials in *~/.pypirc* contains the username/password that twine uses for PyPI.

```
$ cat ~/.pypirc
[distutils]
index-servers =
    pypi

[pypi]
username:<XXXXXX>
password:<XXXXXX>
$ twine upload dist/*
```

4.5.4 Distribute Documentation

Documentation is distributed through readthedocs.org. After a release visit the haka-mqtt readthedocs project, select "Versions" click on "inactive" versions and make sure that the correct versions are marked as "Active".

The `haka-mqtt` project documentation uses PlantUML to draw diagrams and this package is not support out-of-the-box by *readthedocs*. The project root directory contains a `.readthedocs.yml` file to set the build *readthedocs* build environment to one that supports PlantUML and bypass the problem.

4.5.5 Increment Version Number

The release number in *setup.py* has been consumed and should never be used again. Take the time to increment the number, commit the change, then push the change.

```
$ vi setup.py
$ vi doc/source/conf.py
$ git commit setup.py
$ git push origin master
```

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index