HDLController Documentation

Release 0.5.2

Paul-Emmanuel Raoul

Aug 21, 2023

TABLE OF CONTENTS

HDLController is an HDLC controller written in Python and based on the python4yahdlc Python module to encode and decode the HDLC frames.

CHAPTER

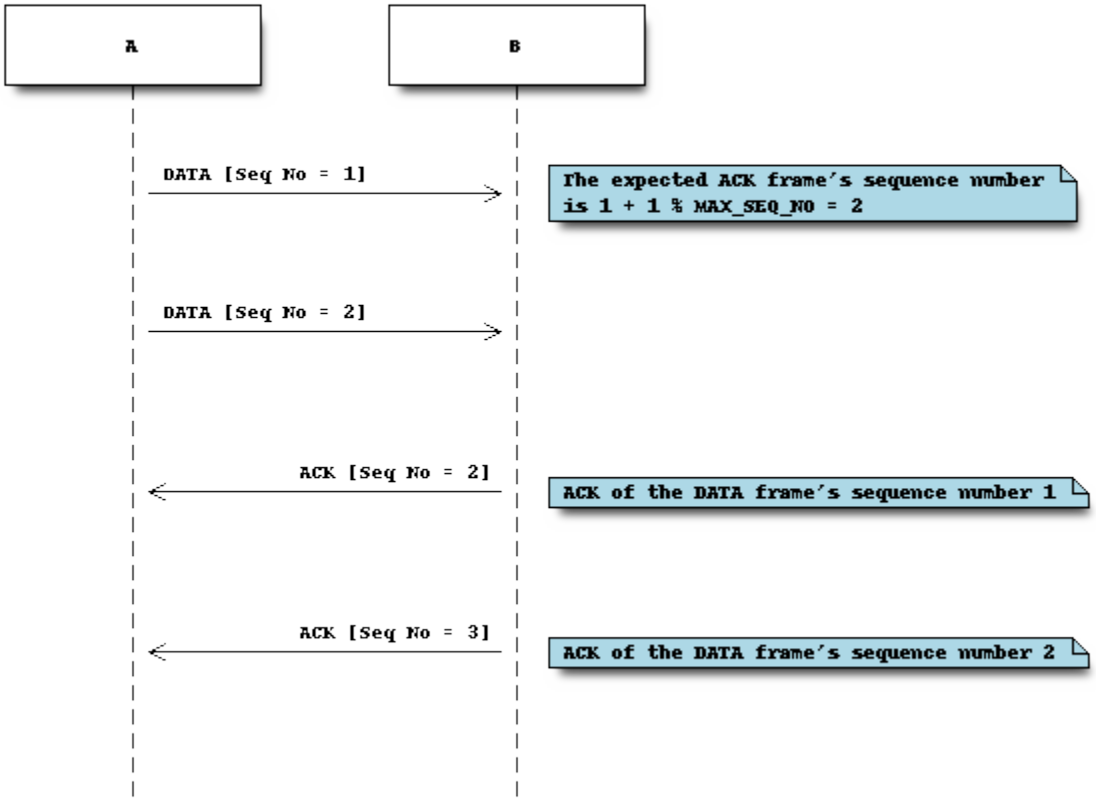
ONE

OVERVIEW

The HDLC controller supports the following frames:

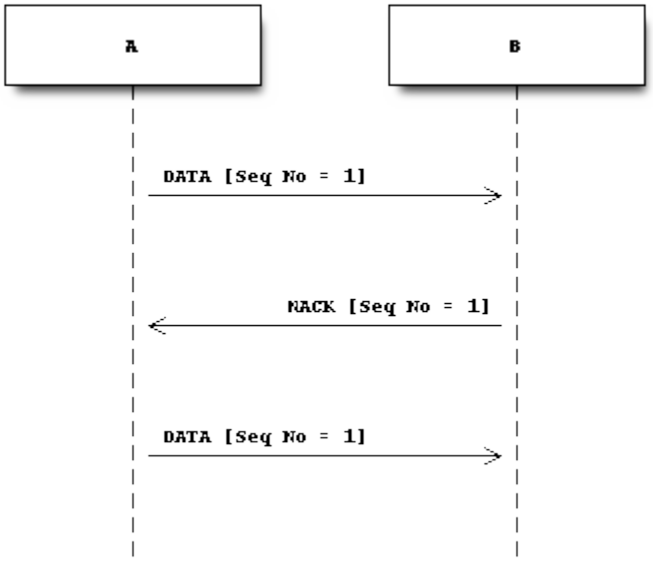
- DATA (I-frame with Poll bit)
- ACK (S-frame Receive Ready with Final bit)
- NACK (S-frame Reject with Final bit)

Each DATA frame must be positively or negatively acknowledged using respectively an ACK or NACK frame. The highest sequence number is 7. As a result, when sending a DATA frame, the expected acknowledgment sequence number is `seq_no + 1 % MAX_SEQ_NO` with `MAX_SEQ_NO = 8`.

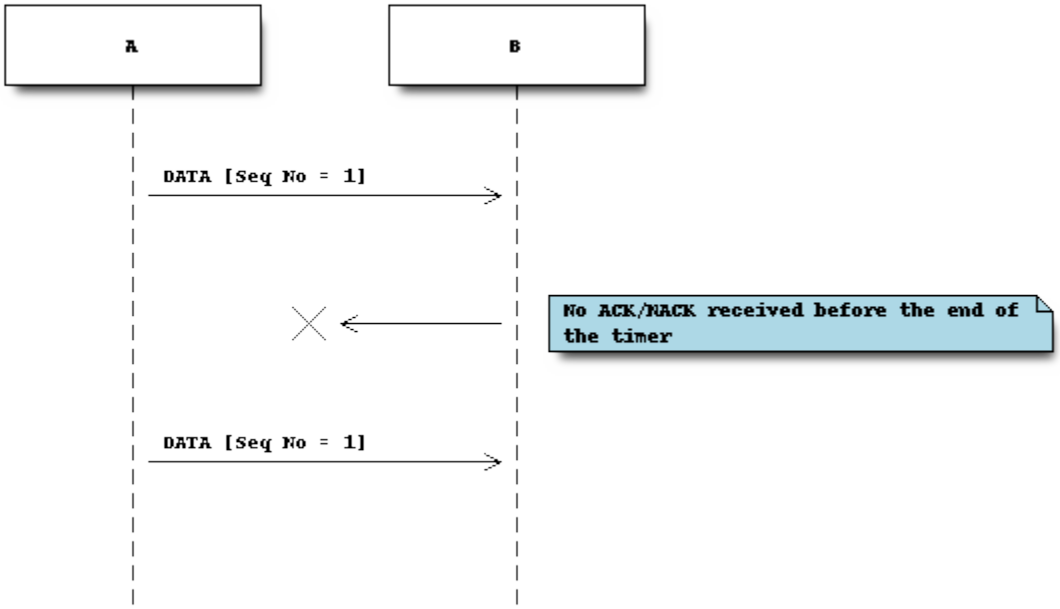


The number of DATA frames that can be sent before receiving the first acknowledgment is determined by the `window` parameter of *`HDLController`*. Its default value is 3.

If the FCS field of a received frame is not valid, an NACK will be sent back with the same sequence number as the one of the corrupted frame to notify the sender about it:



For each DATA frame sent, a timer is started. If the timer ends before receiving any corresponding ACK and NACK frame, the DATA frame will be sent again:



The default timer value is 2 seconds and can be changed using the `sending_timeout` parameter of *HDLController*.

CHAPTER

TWO

INSTALLATION

2.1 From PyPI (recommanded)

```
pip3 install --upgrade hdlcontroller
```

2.2 From sources

HDLController is packaged with Setuptools.

The default Git branch is `develop`. To install the latest stable version, you need to clone the `main` branch.

```
git clone https://github.com/SkypLabs/python-hdlc-controller.git
cd python-hdlc-controller
pip3 install --upgrade .
```

CHAPTER

THREE

USAGE

To create a new HDLC controller instance, you need to call the *HDLController* class with two parameters:

```
hdlc_c = HDLController(read_func, write_func)
```

The first parameter is a function used to read from the serial bus while the second parameter is a function used to write on it. For example, using the pyserial module:

```
ser = serial.Serial('/dev/ttyACM0')

def read_serial():
    return ser.read(ser.in_waiting)

hdlc_c = HDLController(read_serial, ser.write)
```

To start the reception thread:

```
hdlc_c.start()
```

To send a new data frame:

```
hdlc_c.send('Hello world!')
```

And to get the next received data frame available in the *HDLController* internal queue:

```
data = hdlc_c.get_data()
```

The *get_data()* method will block until a new data frame is available.

Finally, to stop all the *HDLController* threads:

```
hdlc_c.stop()
```

CHAPTER

FOUR

MODULES

4.1 HDLController

class hdlcontroller.hdlcontroller.**HDLController**(*read_func: Callable[[], bytes]*, *write_func: Callable[[bytes], Optional[int]]*, *sending_timeout: Timeout = 2.0*, *window: int = 3*, *frames_queue_size: int = 0*, *fcs_nack: bool = True*)

An HDLC controller based on python4yahdlc.

class **Receiver**(*read_func: Callable[[], bytes]*, *write_func: Callable[[bytes], Optional[int]]*, *send_lock: allocate_lock*, *senders_list: Dict[SequenceNumber,* Sender*]*, *frames_received: Queue*, *callback: Optional[Callable[[bytes], None]] = None*, *fcs_nack: bool = True*)

Thread used to receive HDLC frames.

join(*timeout: Optional[Timeout] = None*)

Stops the current thread.

run()

Method representing the thread's activity.

You may override this method in a subclass. The standard run() method invokes the callable object passed to the object's constructor as the target argument, if any, with sequential and keyword arguments taken from the args and kwargs arguments, respectively.

class **Sender**(*write_func: Callable[[bytes], Optional[int]]*, *send_lock: allocate_lock*, *data: bytes*, *seq_no: SequenceNumber*, *timeout: Timeout = 2.0*, *callback: Optional[Callable[[bytes], None]] = None*)

Thread used to send HDLC frames.

ack_received() → None

Informs the sender that the related ACK frame has been received. As a consequence, the current thread is being stopped.

join(*timeout: Optional[Timeout] = None*) → None

Stops the current thread.

nack_received() → None

Informs the sender that an NACK frame has been received. As a consequence, the data frame is being resent.

run() → None

Method representing the thread's activity.

You may override this method in a subclass. The standard run() method invokes the callable object passed to the object's constructor as the target argument, if any, with sequential and keyword arguments taken from the args and kwargs arguments, respectively.

get_data() → bytes

Gets the next frame received.

This method will block until a new data frame is available.

get_senders_number() → int

Returns the number of active senders.

send(*data: bytes*) → None

Sends a new data frame.

This method will block until a new room is available for a new sender. This limit is determined by the size of the window.

set_receive_callback(*callback: Callable[[bytes], None]*) → None

Sets the receive callback function.

This method has to be called before starting the HDLC controller.

set_send_callback(*callback: Callable[[bytes], None]*) → None

Sets the send callback function.

If the HDLC controller has already been started, the new callback function will be taken into account for the next data frames to be sent.

set_sending_timeout(*sending_timeout: Timeout*) → None

Sets the sending timeout.

start() → None

Starts HDLC controller's threads.

stop() → None

Stops HDLC controller's threads.

PYTHON MODULE INDEX

h

INDEX