RawSocketPy Documentation

Release 1.2.1

Alexis Paques

Dec 29, 2021

Contents:

CHAPTER 1

Installation

Listing 1: installation

```
# for development
git clone https://github.com/AlexisTM/rawsocket_python
cd rawsocket_python
sudo python pip . -e

# manually
git clone https://github.com/AlexisTM/rawsocket_python
sudo python setup.py install
```

gevent is an optional dependency that allow concurrency for the RawServer.

You can install it using:

Listing 2: installation

```
sudo -H python -m pip install gevent
```

CHAPTER 2

Quicktest

The simplest example to ensure the library is working for you is to take two machines (or one with two network cards) and run the following.

> Ensure to set **your interface** name instead of **wlp2s0**.

On the first machine:

Listing 1: First machine

```
sudo python -c "from rawsocketpy import RawSocket
sock = RawSocket('wlp2s0', 0xEEFA)
while True: print(sock.recv())"
```

On the second machine:

Listing 2: Second machine

```
sudo python -c "from rawsocketpy import RawSocket; import time
sock = RawSocket('wlp2s0', 0xEEFA)
while True:
  sock.send('Boo')
  print('Boo has been sent')
  time.sleep(0.5)"
```

CHAPTER 3

API

A Raw socket implementation allowing any ethernet type to be used/sniffed.

If gevent is available, sockets are monkey patched and two additionnal asynchronous server implementations are available: `RawAsyncServer`, `RawAsyncServerCallback`

class `rawsocketpy.packet.`**RawPacket**(*data*)

Bases: `object`

RawPacket is the resulting data container of the RawSocket class.

It reads raw data and stores the MAC source, MAC destination, the Ethernet type and the data payload.

RawPacket.success is true if the packet is successfuly read.

__init__(*data*)

A really simple class.

Parameters **data** (*str or bytes or bytearray*) – raw ethernet II frame coming from the socket library, either **bytes in Python3** or **str in Python2**

data = None

Description Payload received

Type str or bytes or bytearray

dest = None

Description Destination MAC address

Type str or bytes or bytearray

src = None

Description Source MAC address

Type str or bytes or bytearray

success = None

Description True if the packet has been successfully unmarshalled

Type bool

type = None

Description Ethertype

Type str or bytes or bytearray

class rawsocketpy.socket.**RawSocket**(*interface*, *protocol*, *sock=None*, *no_recv_protocol=False*)
Bases: object

RawSocket is using the socket library to send raw ethernet frames, using socket.RAW_SOCK

It has a similar API to the socket library: send/recv/close/dup.

BROADCAST = b'\xff\xff\xff\xff\xff\xff'

Description Default MAC address: "\xff\xff\xff\xff\xff\xff"

__init__(*interface*, *protocol*, *sock=None*, *no_recv_protocol=False*)

Parameters

- **interface** (*str*) – interface to be used.
- **protocol** (*int*) – Ethernet II protocol, RawSocket [1536-65535]
- **socket** (*socket.socket*) – Socket to be used (default None), if not given, a new one will be created.
- **no_recv_protocol** (*bool*) – If true (default False), the socket will not subscribe to anything, recv will just block.

dup()
Duplicates the RawSocket

mac = None

Description Source MAC address used for communications - could be modified after initialization

Type str/bytes/bytearray

recv()
Receive data from the socket on the protocol provided in the constructor

Blocks until data arrives. A timeout can be implemented using the socket timeout.

Return type *RawPacket*

send(*msg*, *dest=None*, *ethertype=None*)
Sends data through the socket.

Parameters

- **msg** (*str/bytes/bytearray*) – Payload to be sent
- **dest** (*str/bytes/bytearray*) – recipient such as "\xff\x12\x32\x34\x41" or bytes([1,2,3,4,5,6]). It will broadcast if no given default(None)
- **ethertype** (*str/bytes/bytearray*) – Allow to send data using a different ethertype using the same socket. Default is the protocol given in the constructor.

static sock_create(*interface*, *protocol*, *sock=None*)

class rawsocketpy.server.**RawRequestHandler**(*packet*, *server*)
Bases: object

The class that handles the request. It has access to the packet and the server data.

__init__(*packet*, *server*)
Initialize self. See help(type(self)) for accurate signature.

finish()
empty: To be **overwritten**

handle()
empty: To be **overwritten**

packet = None

Description Packet received

Type *RawPacket*

run()
Run the request handling process:

try:

- self.setup()
- self.handle()

finally:

- self.finish()

server = None

Description Server from which the packet comes

Type *RawServer*

setup()
empty: To be **overwritten**

class rawsocketpy.server.**RawServer**(*interface*, *protocol*, *RequestHandlerClass*)
Bases: object

A **Blocking** base server implementation of a server on top of the RawSocket. It waits for data, encapsulate the data in the RequestHandlerClass provided and blocks until the RequestHandlerClass run() function finishes.

Note packet = recv() -> RequestHandlerClass(packet) -> RequestHandlerClass.run() -> loop

__init__(*interface*, *protocol*, *RequestHandlerClass*)

Parameters

- **interface** (*str*) – interface to be used.
- **protocol** (*int*) – Ethernet II protocol, RawSocket [1536-65535]
- **RequestHandlerClass** (RawServerCallback) – The class that will handle the requests

handle_handler(*handler*)
Manage the handler, can be overwritten

spin()
Loops until self.running becomes False (from a Request Handler or another thread/coroutine)

spin_once()
Handles the next message

class rawsocketpy.server.**RawServerCallback**(*interface*, *protocol*, *RequestHandlerClass*, *callback*)

Bases: *rawsocketpy.server.RawServer*

A blocking server implementation that uses a centralized callback. This is useful for a stateful server.

Note packet = recv() -> RequestHandlerClass(packet, self) -> callback(RequestHandlerClass, self) -> loop

__init__(*interface*, *protocol*, *RequestHandlerClass*, *callback*)

Parameters

- **interface** (*str*) – interface to be used.
- **protocol** (*int*) – Ethernet II protocol, RawSocket [1536-65535]
- **RequestHandlerClass** (RawServerCallback) – The class that will handle the requests
- **callback** (*function*) – callback to be used.

handle_handler(*handler*)
Overwritten: Calls callback(handler, self) instead.

spin()
Loops until self.running becomes False (from a Request Handler or another thread/coroutine)

spin_once()
Handles the next message

rawsocketpy.util.**get_hw**(*ifname*)
Returns a bytearray containing the MAC address of the interface.

Parameters **ifname** (*str*) – Interface name such as wlp2s0

Return type str

Return type bytearray

rawsocketpy.util.**protocol_to_ethertype**(*protocol*)
Convert the int protocol to a two byte chr.

Parameters **protocol** (*int*) – The protocol to be used such as 0x8015

Return type str

rawsocketpy.util.**to_bytes**(**data*)
Flatten the arrays and Converts data to a bytearray

Parameters **data** ([*int*, *bytes*, *bytearray*, *str*, [*int*], [*bytes*], [*bytearray*], [*str*]]) – The data to be converted

Return type bytearray

```
>>> to_bytes("123")
b'123'
>>> to_bytes(1, 2, 3)
b'\x01\x02\x03'
>>> to_bytes("\xff", "\x01\x02")
b'\xff\x01\x02'
>>> to_bytes(1, 2, 3, [4,5,6])
b'\x01\x02\x03\x04\x05\x06'
>>> to_bytes(bytes([1,3,4]), bytearray([6,7,8]), "\xff")
b'\x01\x03\x04\x06\x07\x08\xff'
```

rawsocketpy.util.**to_str**(*data*, *separator=':'*)

Stringify hexadecimal input;

Parameters

- **data** (*str or bytes or bytearray*) – Raw data to print
- **separator** (*str*) – The separator to be used **between** the two digits hexadecimal data.

```
>>> to_str(bytes([1,16,5]))
"01:0F:05"
>>> to_str(bytes([1,16,5]), separator="")
"010F05"
```

CHAPTER 4

Low level socket

You can use the library with a low level socket, where you handle to send and receive.

Listing 1: Sending data

```
from rawsocketpy import RawSocket

# 0xEEFA is the ethertype
# The most common are available here: https://en.wikipedia.org/wiki/EtherType
# The full official list is available here: https://regauth.standards.ieee.org/
↪standards-ra-web/pub/view.html#registries
# Direct link: https://standards.ieee.org/develop/regauth/ethertype/eth.csv
# You can use whatever you want but using a already use type can have unexpected␣
↪behaviour.
sock = RawSocket("wlp2s0", 0xEEFA)

sock.send("some data") # Broadcast "some data" with an ethertype of 0xEEFA

sock.send("personal data", dest="\xAA\xBB\xCC\xDD\xEE\xFF") # Send "personal data to␣
↪\xAA\xBB\xCC\xDD\xEE\xFF with an ether type of 0xEEFA

sock.send("other data", ethertype="\xEE\xFF") # Broadcast "other data" with an ether␣
↪type of 0xEEFF
```

Listing 2: Receiving data

```
from rawsocketpy import RawSocket, to_str

sock = RawSocket("wlp2s0", 0xEEFA)
packet = sock.recv()
# The type of packet is RawPacket() which allows pretty printing and unmarshal the␣
↪raw data.

# If you are using Python2, all data is encoded as unicode strings "\x01.." while␣
↪Python3 uses bytearray.
```

(continues on next page)

(continued from previous page)

```
print(packet) # Pretty print
packet.dest   # string "\xFF\xFF\xFF\xFF\xFF\xFF" or bytearray(b
↪"\xFF\xFF\xFF\xFF\xFF\xFF")
packet.src    # string "\x12\x12\x12\x12\x12\x13" or bytearray(b
↪"\x12\x12\x12\x12\x12\x13")
packet.type   # string "\xEE\xFA" or bytearray([b"\xEE\xFA"]
packegt.data  # string "some data" or bytearray(b"some data"]

print(to_str(packet.dest))      # Human readable MAC:  FF:FF:FF:FF:FF:FF
print(to_str(packet.type, "")) # Human readable type: EEFA
```

CHAPTER 5

Make it a server

Stateless blocking server example: Each time you receive a packet, it will be of type `LongTaskTest` and run `setup()`, `handle()` and finally `finish`. If the handle/setup fails, the `finish` function will be executed.

Listing 1: Blocking Stateless

```
from rawsocketpy import RawServer, RawRequestHandler
import time

class LongTaskTest(RawRequestHandler):
    def setup(self):
        print("Begin")

    def handle(self):
        time.sleep(1)
        print(self.packet)

    def finish(self):
        print("End")

def main():
    rs = RawServer("wlp2s0", 0xEEFA, LongTaskTest)
    rs.spin()

if __name__ == '__main__':
    main()
```

Statefull and **blocking** server using a centralised callback. It does guarantee that the callback is called in ethernet packet order, but if the execution is long, you will loose packets.

Listing 2: Blocking Statefull

```
from rawsocketpy import RawServerCallback, RawRequestHandler
import time
```

(continues on next page)

(continued from previous page)

```
def callback(handler, server):
    print("Testing")
    handler.setup()
    handler.handle()
    handler.finish()

class LongTaskTest(RawRequestHandler):
    def handle(self):
        time.sleep(1)
        print(self.packet)

    def finish(self):
        print("End")

    def setup(self):
        print("Begin")

def main():
    rs = RawServerCallback("wlp2s0", 0xEEFA, LongTaskTest, callback)
    rs.spin()

if __name__ == '__main__':
    main()
```

CHAPTER 6

Go asynchronous

Install gevent and you are ready to use asynchronous servers. The asynchronous server is a very small modification from the base `RawServer`.

Listing 1: Gevent installation command

```
sudo pip -H install gevent
```

Stateless Asynchronous server:

Listing 2: Stateless Asynchronous server

```
from rawsocketpy import RawRequestHandler, RawAsyncServer
import time

class LongTaskTest(RawRequestHandler):
    def handle(self):
        time.sleep(1)
        print(self.packet)

    def finish(self):
        print("End")

    def setup(self):
        print("Begin")

def main():
    rs = RawAsyncServer("wlp2s0", 0xEEFA, LongTaskTest)
    rs.spin()

if __name__ == '__main__':
    main()
```

Statefull Asynchronous server:

Listing 3: Statefull Asynchronous server

```
from rawsocketpy import RawRequestHandler, RawAsyncServerCallback
import time

def callback(handler, server):
    print("Testing")
    handler.setup()
    handler.handle()
    handler.finish()

class LongTaskTest(RawRequestHandler):
    def handle(self):
        time.sleep(1)
        print(self.packet)

    def finish(self):
        print("End")

    def setup(self):
        print("Begin")

def main():
    rs = RawAsyncServerCallback("wlp2s0", 0xEEFA, LongTaskTest, callback)
    rs.spin()

if __name__ == '__main__':
    main()
```

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

r

Index

Symbols

B

D

F

G

H

M

P

R

S

T