WITrace Documentation

Release 1.1.1

Jinghao Shi

Nov 21, 2016

Contents

WlTrace (Wireless Trace) is a Python library that can parse Pcap or Peektagged packet traces. As the name suggests, the main focus is wireless, 802.11 (Wifi) in particular, packet traces.

WlTrace provides a simple abstraction of packet traces, and does all the heavy-lifting behind the scene so that one can focus on more interesting tasks beyond the (tedious) parsing step.

CHAPTER 1

Quick Start

1.1 Installation

You can install this package using `pip`.

```
$ pip install -U wltrace
```

1.2 Usage

```
from wltrace import wltrace

trace = wltrace.load_trace('/path/to/trace')
for pkt in trace:
    # do stuff with pkt
```

CHAPTER 2

Dot11Packet

`wltrace.load_trace` returns a iterator over `Dot11Packet` objects. The following attributes of `Dot11Packet` are available.

2.1 802.11 Attributes

- `type`: int
- `subtype`: int
- Flags (boolean): `to_ds`, `from_ds`, `more_frag`, `retry`, `power`, `more_data`, `protected`, `order`
- `addr1`, `addr2`, `addr3`, `addr4`: these are the MAC address in `11:22:33:aa:bb:cc` format.
- `src`: alias to `addr2`
- `dest`: alias to `addr1`

2.2 Packet Trace Attributes

- `counter`: an integer index of this packet in the trace, starts from 1.
- `ts`, `end_ts`: Python `datetime` object representation the first and last bit of the packet.
- `epoch_ts`, `end_epoch_ts`: POSIX timestamp (float) of the first and last bit of the packet.
- `hash`: a hex digest of the MD5 hash of the packet raw bytes.

2.3 PHY Information

- `phy`: a `PhyInfo` object containing various PHY layer information. See *PHY Information* for details.

2.4 Inferred Information

- `acked`: boolean, whether this packet is acknowledged or not. Only valid for non-broadcast packets.
- `ack_pkt`: reference of the acknowledged packet.

CHAPTER 3

PHY Information

These attributes are available in the `PhyInfo` object.

- `signal`: RSSI in dBm
- `noise`: noise level in dBm
- `freq_mhz`
- `has_fcs`
- `fcs_error`
- `epoch_ts`, `end_epoch_ts`: POSIX timestamp of the first and last bit of the packet
- `rate`: bit rate in Mbps
- `mcs`: MCS index
- `len`: length of the packet, not including any PHY header such as Radiotap header.
- `caplen`: number of bytes actually captured.
- `mactime`: MAC layer TSF count, 64 bit integer.
- `ampdu_ref`: AMPDU reference number if this packet was sent in a AMPDU
- `last_ampdu`: whether this packet was the last packet in the AMPDU.

CHAPTER 4

wltrace package

4.1 Submodules

4.1.1 wltrace.common module

Common interfaces.

class wltrace.common.**GenericHeader**(*fh*, **args*, ***kwargs*)
Bases: object

Base class for general header structure.

This can be a file header, section header (peek-tagged), per-packet header (pcap).

Parameters **fh** (*file object*) – the file handle, which internal pointer points to the start of the header.

FIELDS = None
A list of string representing name of each field in the header, in the order they appear in the PACK_PATTERN format.

It is important that the order of the filed names correspond *strictly* with the order they appear in the header format. If the header has dummy fields, such as padding bytes, you will have to also name them, although you can use the same name for multiple dummy fields.

PACK_PATTERN = None
struct format string used to decode the header bytes.

unpack(*fmt*)

class wltrace.common.**PhyInfo**(**args*, ***kwargs*)
Bases: object

Packet PHY layer information.

PHY information is usually provided in the format of physical layer header, such as Radiotap. PHY information includes:

•signal (int): received RSSI in dBm.

•noise (int): noise level in dBm.

•freq_mhz (int): channel central frequency (MHz)

•has_fcs (bool)

•fcs_error (bool): True if this packet fails the FCS check.

- epoch_ts (float): POSIX timestamp of the first bit of this packet
- end_epoc_ts (float): POSIX timestamp of the last bit of this packet
- rate (float): packet modulation rate (Mbps)
- mcs (int): MCS index (http://mcsindex.com/)
- len (int): packet original length in bytes, including 4 FCS bytes.
- caplen (int): actually stored bytes, probably smaller than `len`.
- mactime (int): MAC layer TSF counter.
- ampdu_ref (int): AMPDU reference number.
- last_frame (bool): True if this packet was the last packet in the AMPDU.

class `wltrace.common.`**`WlTrace`**(*path*, **args*, ***kwargs*)

Bases: `object`

Base class that represents a (wireless) packet trace.

A packet trace is nothing but a sequence of packets. Therefore, the main interface of this object is to yield packet in order. In fact, the object itself is an iterator, which means the packets can only be accessed once in sequence. This is suffice for most purpose, and also reduces memory consumption. Users can always store the packets outside this object if needed.

Parameters **`path`** (*`str`*) – the path of the packet trace file.

Example

This is how `WlTrace` is supposed to be used:

```
cap = WlTrace('path/to/packet/trace.pcap')
for pkt in cap:
  print pkt.counter
```

`next`()

Iteration function.

Note that it is possible to yield dangling ack packets as well, so user can detect if the sniffer missed the previous packet.

`peek`()

Get the current packet without consuming it.

4.1.2 wltrace.dot11 module

IEEE802.11 protocol definitions and utilities.

class `wltrace.dot11.`**`Beacon`**(*pkt*)

Bases: `object`

Payload for 802.11 Beacon packet.

exception `wltrace.dot11.`**`Dot11Exception`**

Bases: `exceptions.Exception`

class wltrace.dot11.**Dot11Packet**(*fh=None*, *phy=None*, *counter=1*, **args*, ***kwargs*)
Bases: *wltrace.common.GenericHeader*

IEEE802.11 packet.

This class parse as much as possible depending on the packet type and subtype.

Parameters

- **fh** (*file object*) – the file's read pointer points to the beginning of a 802.11 packet.
- **phy** (pyparser.capture.common.PhyInfo) – PHY information.
- **counter** (*int*) – packet index in trace file, starting from 1.

FIELDS = ['fc', 'duration', 'addr1']

PACK_PATTERN = '<HH6s'

air_time()
Duration of the packet in air.

Returns duration in seconds.

Return type float

crc_ok

dest
Shortcut to pkt.addr1.

end_epoch_ts

end_ts

epoch_ts

hash

parse_control()

parse_data()

parse_mgmt()

src
Shortcut to pkt.addr2.

ts
Shortcut to pkt.phy.timestamp.

wltrace.dot11.**MAX_ACK_LATENCY_US = 100**
Maximum allowed gap between a packet and its ack in the packet trace.

wltrace.dot11.**is_ack**(*pkt*)
Whether or not the packet is an ack packet.

Parameters **pkt** (*wltrace.dot11.Dot11Packet*) – the packet.

Returns True if it is an ack packet, otherwise False.

Return type bool

wltrace.dot11.**is_beacon**(*pkt*)
Whether a packet is a Beacon packet.

wltrace.dot11.**is_block_ack**(*pkt*)
Whether a packet is a Block Ack packet.

wltrace.dot11.**is_broadcast**(*mac*)
Whether or not a mac is broadcast MAC address.

Parameters **mac** (*str*) – MAC address in string format (xx:xx:xx:xx:xx:xx). Case insensitive.

Returns bool.

wltrace.dot11.**is_highest_rate**(*rate*)
Whether or not the rate is the highest rate (single spatial stream) in rate table.

Parameters **rate** (*int*) – rate in Mbps. Can be 802.11g/n rate.

Returns True if the rate is highest, otherwise False. Note that if rate is not valid, this function returns False, instead of raising an exception.

Return type bool

wltrace.dot11.**is_lowest_rate**(*rate*)
Whether or not the rate is the lowest rate in rate table.

Parameters **rate** (*int*) – rate in Mbps . Can be 802.11g/n rate.

Returns True if the rate is lowest, otherwise False. Note that if rate is not valid, this function returns False, instead of raising an exception.

Return type bool

wltrace.dot11.**is_multicast**(*mac*)
Whether a MAC address is IPV4/V6 multicast address.

See https://en.wikipedia.org/wiki/Multicast_address#Ethernet

Parameters **mac** (*str*) – MAC address

Returns bool

```
>>> is_multicast('01:80:C2:00:00:08')
True
```

wltrace.dot11.**is_qos_data**(*pkt*)
Whether a packet is a QoS Data packet.

wltrace.dot11.**mcs_to_rate**(*mcs*, *bw=20*, *long_gi=True*)
Convert MCS index to rate in Mbps.

See http://mcsindex.com/

Parameters

- **mcs** (*int*) – MCS index
- **bw** (*int*) – bandwidth, 20, 40, 80, ...
- **long_gi** (*bool*) – True if long GI is used.

Returns **rate** – bitrate in Mbps

Return type float

```
>>> mcs_to_rate(5, bw=20, long_gi=False)
57.8
```

```
>>> mcs_to_rate(4, bw=40, long_gi=True)
81
```

```
>>> mcs_to_rate(3, bw=80, long_gi=False)
130
```

```
>>> mcs_to_rate(13, bw=160, long_gi=True)
936
```

wltrace.dot11.**next_seq**(*seq*)
Next sequence number.

Parameters **seq** (*int*) – current sequence number

Returns next sequence number, may wrap around

Return type int

```
>>> next_seq(3)
4
```

```
>>> next_seq(4095)
0
```

wltrace.dot11.**rate_to_mcs**(*rate*, *bw=20*, *long_gi=True*)
Convert bit rate to MCS index.

Parameters

- **rate** (*float*) – bit rate in Mbps
- **bw** (*int*) – bandwidth, 20, 40, 80, ...
- **long_gi** (*bool*) – True if long GI is used.

Returns **mcs** – MCS index

Return type int

```
>>> rate_to_mcs(120, bw=40, long_gi=False)
5
```

4.1.3 wltrace.fusion module

class wltrace.fusion.**Aggregator**(*trace1*, *trace2*, *verbose=False*, **args*, ***kwargs*)
Bases: object

do_aggregate()

wltrace.fusion.**arg_parser**()

wltrace.fusion.**main**()

4.1.4 wltrace.pcap module

Pcap file parser.

class wltrace.pcap.**PcapCapture**(*path*, **args*, ***kwargs*)
Bases: *wltrace.common.WlTrace*

Represent a Pcap packet trace.

Currently only support two link types.

LINKTYPES = [105, 127]

classmethod save(*path*, *pkts*)

exception wltrace.pcap.**PcapException**
Bases: exceptions.Exception

class wltrace.pcap.**PcapHeader**(*fh*, **args*, ***kwargs*)
Bases: *wltrace.common.GenericHeader*

Pcap file header.

The format is documented here: https://wiki.wireshark.org/Development/LibpcapFileFormat

Note that the file header does not contain the total number of packets in the file.

Parameters **fh** (*file object*) – the packet trace file.

FIELDS = ['magic_number', 'version_major', 'version_minor', 'thiszone', 'sigfigs', 'snaplen', 'network']

classmethod to_binary(*endian='@'*, *snaplen=65535*, *network=127*)

class wltrace.pcap.**PcapPacketHeader**(*fh*, *header*, **args*, ***kwargs*)
Bases: *wltrace.common.GenericHeader*

Per packet header in Pcap format.

FIELDS = ['ts_sec', 'ts_usec', 'incl_len', 'orig_len']

classmethod encapsulate(*pkt*, *endian='@'*)

4.1.5 wltrace.peektagged module

Omnipeek peek-tagged packet trace parser.

Peek-tagged file contains several sections:

- `0x7fver`: Omnipeek software version information.
- `sess`: Session information, include total packet number.
- `pkts`: Packets.

Each section starts with a fixed-sized section header, then variable sized payloads. The format is documented here: http://varsanofiev.com/inside/airopeekv9.htm

class wltrace.peektagged.**PeektaggedCapture**(*path*, **args*, ***kwargs*)
Bases: *wltrace.common.WlTrace*

Peek-tagged capture file.

Here we know the total number of packets beforehand from the "sess" section. So this class has an extra `total_packets` attribute.

exception wltrace.peektagged.**PeektaggedException**
Bases: exceptions.Exception

class wltrace.peektagged.**PeektaggedPacketHeader**(*fh*, **args*, ***kwargs*)
Bases: object

Per packet header.

This is peek-tagged format's specific way to convey PHY layer information in packet trace.

TAGS = {0: ('len', '<I'), 1: ('ts_low', '<I'), 2: ('ts_high', '<I'), 3: ('flags', '<I'), 4: ('channel', '<I'), 5: ('rate', '<I'), 6: ('sig

to_phy()
Convert this to the standard pyparser.capture.common.PhyInfo class.

class wltrace.peektagged.**PeektaggedSectionHeader**(*fh*, **args*, ***kwargs*)
Bases: *wltrace.common.GenericHeader*

Peek-tagged section header.

Parameters

- **fh** (*file object*) – file to be read.
- **load_payload** (*bool*) – whether or not read the payload from the file. By default, it is False. For small sections, such as "0x7fver", "sess", it is no big deal. But for "pkts" section, which can be huge, we do not want to load the entire section at once.

FIELDS = ['tag', 'len', 'pad']

PACK_PATTERN = '<4sII'

4.1.6 wltrace.quality module

class wltrace.quality.**CaptureQuality**(*cap*, *ta*, *ra*, **args*, ***kwargs*)
Bases: object

missing_ack_count

missing_tx_count

4.1.7 wltrace.radiotap module

Radiotap header parser.

class wltrace.radiotap.**RadiotapHeader**(*fh*, **args*, ***kwargs*)
Bases: *wltrace.common.GenericHeader*

Radiotap header.

See this document for radiotap header format: http://www.radiotap.org/

See this document for all defined radiotap fields: http://www.radiotap.org/defined-fields/all

FIELDS = ['_it_version', '_it_pad', '_it_len', '_it_present']

PACK_PATTERN = '<BBHI'
Radiotap header is always in little endian.

PRESENT_FLAGS = [(0, 'Q', 'mactime', 8), (1, 'B', '_flags', 1), (2, 'B', 'rate', 1), (3, 'I', '_channel', 2), (4, 'xx', 'unused', 1

classmethod from_phy_info(*phy*)

to_binary()

to_phy()

4.1.8 wltrace.utils module

Various utilitis for packet trace parsing.

wltrace.utils.**align_up**(*offset*, *align*)
Align offset up to align boundary.

Parameters

- **offset** (*int*) – value to be aligned.
- **align** (*int*) – alignment boundary.

Returns aligned offset.

Return type int

```
>>> align_up(3, 2)
4
```

```
>>> align_up(3, 1)
3
```

wltrace.utils.**bin_to_mac**(*bin*, *size=6*)

Convert 6 bytes into a MAC string.

Parameters **bin** (*str*) – hex string of lenth 6.

Returns String representation of the MAC address in lower case.

Return type str

Raises Exception – if len(bin) is not 6.

wltrace.utils.**calc_padding**(*fmt*, *align*)

Calculate how many padding bytes needed for fmt to be aligned to align.

Parameters

- **fmt** (*str*) – struct format.
- **align** (*int*) – alignment (2, 4, 8, etc.)

Returns padding format (e.g., various number of 'x').

Return type str

```
>>> calc_padding('b', 2)
'x'
```

```
>>> calc_padding('b', 3)
'xx'
```

wltrace.utils.**packet_gap**(*first*, *second*)

wltrace.utils.**pairwise**(*it*)

wltrace.utils.**win_ts**(*high*, *low*)

Convert Windows timestamp to Unix timestamp.

Windows timestamp is a 64-bit integer, the value of which is the number of 100 ns intervals from 1/1/1601-UTC.

Parameters

- **high** (*int*) – high 32 bits of windows timestamp.
- **low** (*int*) – low 32 bits of windows timestamp.

Returns Python timestamp (datetime.datetime object).

wltrace.utils.**win_ts_to_unix_epoch**(*high*, *low*)
Convert Windows timestamp to POSIX timestamp.

See https://goo.gl/VVX0nk

Parameters

- **high** (*int*) – high 32 bits of windows timestamp.
- **low** (*int*) – low 32 bits of windows timestamp.

Returns float

4.1.9 wltrace.wltrace module

Wireless Packet Trace

This module can load a packet trace, and yields a sequence of packets. Currently, only IEEE 802.11 (aka Wifi) pakcet traces saved in Pcap or Omnipeek's peek-tagged format are supported. For Pcap format, this module can parse the Radiotap header if exists.

wltrace.wltrace.**FILE_TYPE_HANDLER = {'\xd4\xc3\xb2\xa1': <class 'wltrace.pcap.PcapCapture'>, '\xa1\xb2\xc3\xd4':**
A map from magic bytes to file handler.

wltrace.wltrace.**MAGIC_LEN = 4**
File type magic length in bytes.

wltrace.wltrace.**is_packet_trace**(*path*)
Determine if a file is a packet trace that is supported by this module.

Parameters **path** (*str*) – path to the trace file.

Returns True if the file is a valid packet trace.

Return type bool

wltrace.wltrace.**load_trace**(*path*, **args*, ***kwargs*)
Read a packet trace file, return a *wltrace.common.WlTrace* object.

This function first reads the file's magic (first FILE_TYPE_HANDLER bytes), and automatically determine the file type, and call appropriate handler to process the file.

Parameters **path** (*str*) – the file's path to be loaded.

Returns WlTrace object.

4.2 Module contents

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

W

R

S

T

U

W