
disspcap Documentation

Release 0.0.1

Daniel Uhricek

Apr 15, 2019

1	Requirements	3
1.1	Build dependencies	3
1.2	Python dependencies	3
2	Build and install	5
2.1	Install build requirements	5
2.2	C++ shared library	5
2.3	Python package	5
3	C++	7
3.1	Basics	7
4	Python	9
4.1	Basics	9
4.2	Examples	9
4.2.1	Simple statistics	9
4.2.2	DNS	10
5	C++ API	13
5.1	Pcap	13
5.2	Packet	13
5.3	Ethernet	14
5.4	IPv4	14
5.5	IPv6	15
5.6	UDP	15
5.7	TCP	15
5.8	DNS	16
6	Python API	17
6.1	Pcap	17
6.2	Packet	17
6.3	Ethernet	18
6.4	IPv4	18
6.5	IPv6	18
6.6	UDP	19
6.7	TCP	19
6.8	DNS	19

7	Contribute	21
7.1	How to contribute	21
8	License	23
9	Contact	25

Disspcap is a minimalist library for packet examination implemented in C++ and with available binding to Python. Attempting to be *simple* and *fast*. Disspcap provides simple alternative to robust pcap-related libraries and frameworks.

CHAPTER 1

Requirements

Note: Disspcap is currently for Linux based platforms only.

1.1 Build dependencies

- C++ compiler supporting c++11
- libpcap-dev package

1.2 Python dependencies

- pybind11 >= 2.2

CHAPTER 2

Build and install

2.1 Install build requirements

```
$ sudo apt-get install libpcap-dev
```

2.2 C++ shared library

```
$ git clone https://github.com/danieluhricek/disspcap
$ cd disspcap
$ make
```

2.3 Python package

```
$ pip install disspcap
```

or

```
$ git clone https://github.com/danieluhricek/disspcap
$ cd disspcap
$ python setup.py install
```


3.1 Basics

```
#include <disspcap/pcap.h>
#include <disspcap/packet.h>
#include <iostream>

using namespace disspcap;

int main(int argc, char* argv[])
{
    Pcap pcap("path_to_pcap");

    auto packet = pcap.next_packet();

    if (packet->ethernet()) {
        std::cout << packet->ethernet()->source() << std::endl;
        std::cout << packet->ethernet()->destination() << std::endl;
        std::cout << packet->ethernet()->type() << std::endl;
    }

    if (packet->ipv4()) {
        std::cout << packet->ipv4()->source() << std::endl;
        std::cout << packet->ipv4()->destination() << std::endl;
        std::cout << packet->ipv4()->protocol() << std::endl;
    }

    if (packet->ipv6()) {
        std::cout << packet->ipv6()->source() << std::endl;
        std::cout << packet->ipv6()->destination() << std::endl;
        std::cout << packet->ipv6()->next_header() << std::endl;
    }
}
```

(continues on next page)

(continued from previous page)

```
    if (packet->udp()) {
        std::cout << packet->udp()->source_port() << std::endl;
        std::cout << packet->udp()->destination_port() << std::endl;
    }

    if (packet->tcp()) {
        std::cout << packet->tcp()->source_port() << std::endl;
        std::cout << packet->tcp()->destination_port() << std::endl;
    }

    return 0;
}
```

4.1 Basics

```
>>> import disspcap
>>> pcap = disspcap.Pcap('path_to_pcap')
>>> packet = pcap.next_packet()
```

Now we can inspect packet.

```
>>> packet.ethernet.source
73:15:B8:A6:58:73
>>> packet.ethernet.type
IPv4
>>> packet.ipv4.destination
105.190.108.167
>>> packet.ipv4.protocol
TCP
>>> packet.tcp.destination_port
22
```

4.2 Examples

4.2.1 Simple statistics

```
import disspcap

ethernet_packets = 0
ipv4_packets = 0
ipv6_packets = 0
tcp_packets = 0
```

(continues on next page)

(continued from previous page)

```
udp_packets = 0

pcap = disspcap.Pcap('path_to_pcap')
packet = pcap.next_packet()

while packet:
    if (packet.ethernet):
        ethernet_packets += 1

    if (packet.ipv4):
        ipv4_packets += 1

    if (packet.ipv6):
        ipv6_packets += 1

    if (packet.udp):
        udp_packets += 1

    if (packet.tcp):
        tcp_packets += 1

    packet = pcap.next_packet()

print(f'Number of ethernet packets {ethernet_packets}')
print(f'Number of ipv4 packets {ipv4_packets}')
print(f'Number of ipv6 packets {ipv6_packets}')
print(f'Number of udp packets {udp_packets}')
print(f'Number of tcp packets {tcp_packets}')
```

4.2.2 DNS

```
import disspcap

i = 1
pcap = disspcap.Pcap('path_to_pcap')
packet = pcap.next_packet()

while packet:
    if packet.dns:
        if packet.dns.qr == 1:
            print(f'\nPacket #{i}:')

            print('  Answers: ')
            for ans in packet.dns.answers:
                print(f'    {ans}')

            print('  Authoritatives: ')
            for auth in packet.dns.authoritatives:
                print(f'    {auth}')

            print('  Additionals: ')
            for add in packet.dns.additionals:
                print(f'    {add}')
```

(continues on next page)

(continued from previous page)

```
i += 1
packet = pcap.next_packet()
```


5.1 Pcap

class Pcap

Holds pcap file information and provides methods for pcap manipulation.

Pcap()

Default constructor of a new Pcap::Pcap object. Needs opening afterwards.

Pcap(const std::string& filename)

Constructs Pcap objects, opens pcap file and initializes data.

Parameters **file_name** – Path to pcap.

void open_pcap(const std::string& filename)

Opens pcap. Only needed if Pcap object created with default constructor.

Parameters **file_name** – Path to pcap.

std::unique_ptr<Packet> next_packet()

Read next packet from a pcap file. Returns nullptr if no more packets.

Returns Next *Packet* parsed out of pcap file.

5.2 Packet

class Packet

Packet(uint8_t* data, unsigned int length)

Constructor of a new Packet *Packet* object.

Parameters

- **data** – Pointer to start of pcap bytes.

- `length` – Length of read packet.

`const Ethernet* ethernet() const`

Returns *Ethernet* object or `nullptr`.

`const IPv4* ipv4() const`

Returns *IPv4* object or `nullptr`.

`const IPv6* ipv6() const`

Returns *IPv6* object or `nullptr`.

`const UDP* udp() const`

Returns *UDP* object or `nullptr`.

`const TCP* tcp() const`

Returns *TCP* object or `nullptr`.

`const DNS* dns() const`

Returns *DNS* object or `nullptr`.

`unsigned int length() const`

Returns Packet length.

`unsigned int payload_length() const`

Returns Payload length (packet data following transport protocols).

`uint8_t* payload()`

Returns Payload data

5.3 Ethernet

`class Ethernet`

`const std::string& source() const`

Returns Source MAC address. (e.g. "54:75:d0:c9:0b:81")

`const std::string& destination() const`

Destination Source MAC address. (e.g. "54:75:d0:c9:0b:81")

`const std::string& type() const`

Returns "IPv4", "IPv6" or "ARP"

5.4 IPv4

`class IPv4`

`const std::string& source() const`

Returns Source IPv4 address. (e.g. "192.168.0.1")

```
const std::string& destination() const
    Returns Destination IPv4 address. (e.g. "192.168.0.1")
const std::string& protocol() const
    Returns Next protocol. (e.g., "TCP", "UDP", "ICMP"...)
```

```
const std::string& header_length() const
    Returns IPv4 header length.
```

5.5 IPv6

```
class IPv6
```

```
const std::string& source() const
    Returns Source IPv6 address. (e.g. "fe80::0202:b3ff:fe1e:8329")
const std::string& destination() const
    Returns Destination IPv6 address. (e.g. "fe80::0202:b3ff:fe1e:8329")
const std::string& next_header() const
    Returns Next header type. (e.g., "TCP", "UDP", "ICMP"...)
```

5.6 UDP

```
class UDP
```

```
unsigned int source_port() const
    Returns Source port number.
unsigned int destination_port() const
    Returns Destination port number.
```

5.7 TCP

```
class TCP
```

```
unsigned int source_port() const
    Returns Source port number.
unsigned int destination_port() const
    Returns Destination port number.
```

5.8 DNS

class DNS

unsigned int qr() **const**

Returns 0 (Query) or 1 (Response).

unsigned int question_count() **const**

Returns Number of question entries.

unsigned int answer_count() **const**

Returns Number of answer entries.

unsigned int authority_count() **const**

Returns Number of entries in authoritative NS section.

unsigned int additional_count() **const**

Returns Number of additional resource records.

const std::vector<std::string>& answers() **const**

Returns Answer RRs. Vector of std::string formatted as: "google.com A 172.217.23.206"

const std::vector<std::string>& authoritatives() **const**

Returns Authoritative NS RRs. Vector of std::string formatted as: "google.com NS ns4.google.com"

const std::vector<std::string>& additionals() **const**

Returns Additional RRs. Vector of std::string formatted as: "google.com A 172.217.23.206"

6.1 Pcap

class Pcap

Holds pcap file information and provides methods for pcap manipulation.

__init__ (*file*)

Parameters **file** – Path to pcap.

next_packet ()

Returns Next *Packet* parsed out of pcap file.

6.2 Packet

class Packet

ethernet

Ethernet object or None.

ipv4

IPv4 object or None.

ipv6

IPv6 object or None.

udp

UDP object or None.

tcp

TCP object or None.

dns

DNS object or None.

payload_length

Length of payload transport protocol.

payload

Payload of `bytes` following transport protocol.

6.3 Ethernet

class Ethernet

source

Source MAC address. (e.g. `'54:75:d0:c9:0b:81'`)

destination

Destination MAC address. (e.g. `'54:75:d0:c9:0b:81'`)

type

`'IPv4'`, `'IPv6'` or `'ARP'`

6.4 IPv4

class IPv4

source

Source IPv4 address. (e.g. `'192.168.0.1'`)

destination

Destination IPv4 address. (e.g. `'192.168.0.1'`)

protocol

Next protocol. (e.g. `'TCP'`, `'UDP'`, `'IGMP'...`)

header_length

IPv4 header length.

6.5 IPv6

class IPv6

source

Source IPv6 address. (e.g. `'fe80::0202:b3ff:fe1e:8329'`)

destination

Destination IPv6 address. (e.g. `'fe80::0202:b3ff:fe1e:8329'`)

next_header

Next header type. (e.g. `'TCP'`, `'UDP'`, `'IGMP'...`)

6.6 UDP

class UDP

source_port

Source port number.

destination_port

Destination port number.

6.7 TCP

class TCP

source_port

Source port number.

destination_port

Destination port number.

6.8 DNS

class DNS

qr

0 (Query) or 1 (Response).

question_count

Number of question entries.

answer_count

Number of answer entries.

authority_count

Number of entries in authoritative NS section.

additional_count

Number of additional resource records.

answers

Answer RRs. List of strings formatted as: ['google.com A 172.217.23.206', ...]

authoritatives

Authoritative NS RRs. List of strings formatted as: ['google.com NS ns4.google.com', ...]
]

additional

Additional RRs. List of strings formatted as: ['google.com A 172.217.23.206', ...]

Disspcap is a new project and is open for contributions. Main repository is at: <https://github.com/danieluhricek/disspcap>

7.1 How to contribute

- [Create an issue](#) for found bugs.
- Implement dissecting of any other application protocol.
- Implement other link-layer protocol parsing.

CHAPTER 8

License

MIT License

Copyright (c) 2018 Daniel Uhřiček

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

CHAPTER 9

Contact

Daniel Uhricek

- daniel.uhricek@gypri.cz
- [LinkedIn](#)
- [Github](#)

Symbols

`__init__()` (*Pcap method*), 17

A

`additional_count` (*DNS attribute*), 19

`additional`s (*DNS attribute*), 19

`answer_count` (*DNS attribute*), 19

`answers` (*DNS attribute*), 19

`authoritat`ives (*DNS attribute*), 19

`author`ity_count (*DNS attribute*), 19

D

`destination` (*Ethernet attribute*), 18

`destination` (*IPv4 attribute*), 18

`destination` (*IPv6 attribute*), 18

`destination_port` (*TCP attribute*), 19

`destination_port` (*UDP attribute*), 19

`DNS` (*built-in class*), 16, 19

`dns` (*Packet attribute*), 17

E

`Ethernet` (*built-in class*), 14, 18

`ethernet` (*Packet attribute*), 17

H

`header_length` (*IPv4 attribute*), 18

I

`IPv4` (*built-in class*), 14, 18

`ipv4` (*Packet attribute*), 17

`IPv6` (*built-in class*), 15, 18

`ipv6` (*Packet attribute*), 17

N

`next_header` (*IPv6 attribute*), 18

`next_packet()` (*Pcap method*), 17

P

`Packet` (*built-in class*), 13, 17

`Packet()` (*Packet method*), 13

`payload` (*Packet attribute*), 18

`payload_length` (*Packet attribute*), 17

`Pcap` (*built-in class*), 13, 17

`Pcap()` (*Pcap method*), 13

`protocol` (*IPv4 attribute*), 18

Q

`qr` (*DNS attribute*), 19

`question_count` (*DNS attribute*), 19

S

`source` (*Ethernet attribute*), 18

`source` (*IPv4 attribute*), 18

`source` (*IPv6 attribute*), 18

`source_port` (*TCP attribute*), 19

`source_port` (*UDP attribute*), 19

T

`TCP` (*built-in class*), 15, 19

`tcp` (*Packet attribute*), 17

`type` (*Ethernet attribute*), 18

U

`UDP` (*built-in class*), 15, 19

`udp` (*Packet attribute*), 17