PyANCP Documentation

Release 0.1

Christian Giese

Aug 29, 2017

Contents

Python ANCP (RFC 6320) client and library.

PyANCP requires Python 2.7 or later, or Python 3.2 or later.

State: **BETA**

ANCP Library Example:

```
from ancp.client import Client
from ancp.subscriber import Subscriber

# setup ancp session
client = Client(address="1.2.3.4")
if client.connect():
    # create ancp subscribers
    S1 = Subscriber(aci="0.0.0.0 eth 1", up=1024, down=16000)
    S2 = Subscriber(aci="0.0.0.0 eth 2", up=2048, down=32000)
    # send port-up for ancp subscribers
    client.port_up([S1, S2])
    # keep session active
    try:
        while client.established.is_set():
            time.sleep(1)
    except KeyboardInterrupt:
        # send port-down for ancp subscribers
        client.port_down([S1, S2])
        client.disconnect()
```

Contents:

CHAPTER 1

ANCP Client

Currently there is just an example (*bin/client.py*) which shows how to use the client library. An ANCP client is planned for future releases.

CHAPTER 2

ANCP Client Library

The client library allows to setup and control sessions to an ANCP server.

Session Setup

Following an example of how to create an ANCP session.

```
from ancp.client import Client

client = Client(address="1.2.3.4")
client.connect()
```

Warning: IPv6 is currently not supported!

It is also possible to specify the source address and destination port (default 6068). The default tech type is *DSL* which can be changed to *ANY* or *PON*. The argument *timer* (default 25 seconds) specifies the interval of periodically adjacency messages to monitor the session.

```
from ancp.client import Client
from ancp.client import TechTypes

client = Client(address="1.2.3.4", source_address="1.2.3.5", port=6068
                tech_type=TechTypes.DSL, timer=25.0)
client.connect()
```

The *connect* method creates a TCP session and starts a background thread which responds to messages from the server and generates periodically adjacency messages (*timer*).

Following an example which shows how to keep a session active until *KeyboardInterrupt*.

```
try:
    while client.established.is_set():
        time.sleep(1)
except KeyboardInterrupt:
    client.disconnect()
```

The *disconnect* method send an *ANCP RSTACK* message to the server, waits up to 1 seconds for response and closes TCP session.

ANCP Subscriber

ANCP subscribers are requires to generate *Port Up/Down Messages*.

```
from ancp.subscriber import Subscriber

S1 = Subscriber(aci="0.0.0.0 eth 1", up=1024, down=16000)
```

All supported line attributes are described in *ancp/subscriber.py*. The argument *aci* is mandatory. Attributes can be updated (e.g. *S1.up=1000*) or removed (e.g. *S1.up=None*).

Port Up/Down Messages

It is possible to send multiple port up/down in a single TCP message.

```
# create ancp subscribers
S1 = Subscriber(aci="0.0.0.0 eth 1", up=1024, down=16000)
S2 = Subscriber(aci="0.0.0.0 eth 2", up=2048, down=32000)
S3 = Subscriber(aci="0.0.0.0 eth 3", up=2048, down=32000)

# send single port up message
client.port_up(S1)

# send multiple port up in a single tcp message
client.port_up([S2, S3])
```

The *port_down* method behaves similar to *port_up*.

It is also possible to update line attributes without sending a port down message.

```
# create ancp subscribers
S1 = Subscriber(aci="0.0.0.0 eth 1", up=1024, down=16000)

# send single port up message
client.port_up(S1)

# change line attributes and send port up
S1.up=768
S1.down=14000
client.port_up(S1)

# send port up again
client.port_up(S1)
```

CHAPTER 3

Code Documentation

ancp

Python ANCP Client

ancp/client.py

ANCP Client

Copyright 2017 Christian Giese <cgiese@juniper.net>

class ancp.client.**AdjacencyState**

ESTAB = 4

IDLE = 1

SYNRCVD = 3

SYNSENT = 2

class ancp.client.**Capabilities**

OAM = 4

TOPO = 1

class ancp.client.**Client**(*address*, *port=6068*, *tech_type=5*, *timer=25.0*, *source_address=None*)

ANCP Client

Parameters

- **address** (*str*) – ANCP server address (IPv4)
- **port** (*int*) – ANCP port (default: 6086)

- **tech_type** (ancp.client.TechTypes) – tech type (default=DSL)
- **timer** (*int*) – adjacency timer (default=25.0)
- **source_address** (*str*) – optional source address

connect()

disconnect(*send_ack=False*)

port_down(*subscribers*)
send port-down message

For backwards compability single value ANCP subscribers are accepted.

Parameters **subscriber** ([ancp.subscriber.Subscriber]) – collection of ANCP subscribers

port_up(*subscribers*)
send port-up message

For backwards compability single value ANCP subscribers are accepted.

Parameters **subscriber** ([ancp.subscriber.Subscriber]) – collection of ANCP subscribers

class ancp.client.**MessageCode**

ACK = 3

RSTACK = 4

SYN = 1

SYNACK = 2

class ancp.client.**MessageType**

ADJACENCY = 10

ADJACENCY_UPDATE = 85

PORT_DOWN = 81

PORT_MANAGEMENT = 32

PORT_UP = 80

class ancp.client.**ResultCodes**

NoResult = 0

class ancp.client.**ResultFields**

AckAll = 2

Failure = 4

Ignore = 0

Nack = 1

Success = 3

class ancp.client.**TechTypes**

ANY = 0

DSL = 5

PON = 1

ancp.client.**tomac**(*v*)
Tuple to MAC Address

Parameters **v** (*tuple*) – MAC address

Returns MAC address

Return type str

ancp/subscriber.py

ANCP Subscribers

class ancp.subscriber.**DataLink**
Access-Loop-Encapsulation - Data Link

ATM_AAL5 = 0

ETHERNET = 1

class ancp.subscriber.**DslType**
DSL Types

ADSL = 1

ADSL2 = 2

ADSL2P = 3

OTHER = 0

SDSL = 6

VDSL1 = 4

VDSL2 = 5

class ancp.subscriber.**Encap1**
Access-Loop-Encapsulation - Encapsulation 1

DOUBLE_TAGGED_ETHERNET = 3

NA = 0

SINGLE_TAGGED_ETHERNET = 2

UNTAGGED_ETHERNET = 1

class ancp.subscriber.**Encap2**
Access-Loop-Encapsulation - Encapsulation 2

EOAAL5_LLC = 6

EOAAL5_LLC_FCS = 5

EOAAL5_NULL = 8

EOAAL5_NULL_FCS = 7

IPOA_LLC = 3

IPOA_Null = 4

PPPOA_LLC = 1

PPPOA_NULL = 2

class ancp.subscriber.**LineState**
Line States

IDLE = 2

SHOWTIME = 1

SILENT = 3

class ancp.subscriber.**Subscriber**(*aci*, ***kwargs*)
ANCP Subscriber

Parameters

- **aci** (*str*) – Access-Loop-Circuit-ID
- **ari** (*str*) – Access-Loop-Remote-ID
- **aaci_bin** (*int or tuple*) – Access-Aggregation-Circuit-ID-Binary
- **aaci_ascii** (*str*) – Access-Aggregation-Circuit-ID-ASCII
- **state** (ancp.subscriber.LineState) – DSL-Line-State
- **up** (*int*) – Actual-Net-Data-Rate-Upstream
- **down** (*int*) – Actual-Net-Data-Rate-Downstream
- **min_up** (*int*) – Minimum-Net-Data-Rate-Upstream
- **min_down** (*int*) – Minimum-Net-Data-Rate-Downstream
- **att_up** (*int*) – Attainable-Net-Data-Rate-Upstream
- **att_down** (*int*) – Attainable-Net-Data-Rate-Downstream
- **max_up** (*int*) – Maximum-Net-Data-Rate-Upstream
- **max_down** (*int*) – Maximum-Net-Data-Rate-Downstream
- **dsl_type** (ancp.subscriber.DslType) – DSL-Type
- **data_link** (ancp.subscriber.DataLink) – Access-Loop-Encapsulation - Data Link
- **encap1** (ancp.subscriber.Encap1) – Access-Loop-Encapsulation - Encapsulation 1
- **encap2** (ancp.subscriber.Encap2) – Access-Loop-Encapsulation - Encapsulation 2

aaci_bin

tlvs

class ancp.subscriber.**TLV**(*t*, *val*)

len

off

type

val

class ancp.subscriber.**TlvType**
TLV Types

AACI_ASCII = 3

AACI_BIN = 6

ACC_LOOP_ENC = 144

ACI = 1

ARI = 2

ATT_DOWN = 134

ATT_UP = 133

DOWN = 130

LINE = 4

MAX_DOWN = 136

MAX_UP = 135

MIN_DOWN = 132

MIN_UP = 131

STATE = 143

TYPE = 145

UP = 129

ancp.subscriber.**access_loop_enc**(*data_link*, *encap1*, *encap2*)
Create the Access Loop Tlv

Parameters

- **data_link** (ancp.subscriber.DataLink) – The Data link type
- **encap1** (ancp.subscriber.Encap1) – The first Encapsulation type
- **encap2** (ancp.subscriber.Encap2) – The second Encapsulation type

Return type *TLV*

ancp.subscriber.**mktlvs**(*tlvs*)

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

Index