netengine Documentation

Release 0.1

Alessandro Bucciarelli, Federico Capoano

January 19, 2017

Contents

Netengine is a python library that aims to provide a single API to extract common information from network devices using different protocols (eg: SNMP, SSH, HTTP) and different firwmares (eg: OpenWRT, AirOS).

You can immagine **Netengine** as a read-only ORM (Object Relational Mapper) equivalent for networks.

CHAPTER 1

Motivations

While dealing with networks in the real world, it's highly probable that you will deal with a network which is made with very different routers, switches and servers. Some may support standard SNMP mibs, some may not, some may implement other HTTP APIs, some may even implement obscure/custom SNMP mibs.

If you need to develop a web application that automates some networking tasks, you don't want to deal with all those differences in the application code, because it would become hard to mantain very soon. You also might not want to tie your web app code to a specific vendor or firmware because that would make your software unflexible.

If we had a single API we could let web developers focus on the task they need to accomplish rather than dealing with SNMP, SSH, different firmwares, different linux distributions and so on.

The goal of this project is to build that single API.

CHAPTER 2

Status of this project

We are currently in 0.1 beta version.

The 0.1 final version will be out by April 2015.

CHAPTER 3

Install

Install the development version (tarball):

```
pip install https://github.com/ninuxorg/netengine/tarball/master
```

Alternatively, you can install via pip using git:

```
pip install -e git+git://github.com/ninuxorg/netengine#egg=netengine
```

CHAPTER 4

Contents:

4.1 Usage

The usage of Netengine module requires it to be installed properly as explained in index_reference. If you have an installation under a virtualenv, enter the folder /bin and type:: source activate otherwise (if you have installed globally) just open an editor as bpython and you we are ready to go.

These are the main steps to follow to use the module:

- import the correct backend and supported framework
- declare a device using the proper constructor
- invoke methods over the device just declared

So we have:

```
from netengine.backends.<backend_name> import <supported_firmware>

<device_name> = supported_firmware_constructor
```

To invoke methods over the just declared device it's necessary to use the dot notation as:

```
<device_name>.<method or property>
```

Further example will be found inside dedicated docs for every backend

4.2 Running tests

Install nose:

```
pip install nose
```

Clone repo:

```
git clone git://github.com/ninuxorg/netengine

cd netengine/
```

Edit settings json file according to your network:

```
cp test-settings.example.json test-settings.json
vim test-settings.json
```

Run tests with:

```
nosetests
```

See test coverage with:

```
nosetests --with-coverage --cover-package=netengine
```

Run specific tests by specifying the relative path:

```
# base tests
nosetests tests.base

# snmp tests
nosetests tests.snmp
# snmp openwrt specific tests
nosetests tests.snmp.openwrt

# ssh tests
nosetests tests.ssh
# ssh airos specific tests
nosetests tests.ssh.airos
```

4.3 SSH backend

4.3.1 SSH

SSH (Secure SHell) is a network protocol which can be used to remotely log in a user into a device, it provides cryptographic communication between two networked hosts.

This backend provides access to remote device through SSH (Secure SHell). Moreover SSH backend supports two main firmwares:

- AirOS
- OpenWRT

4.3.2 AirOS example

Here it is an example of how to import the SSH.AirOS backend, declare a device and invoke methods and properties on it:

```
from netengine.backends.ssh import AirOS
```

Now we can invoke methods and properties by simply typing:

```
device = AirOS('10.40.0.1', 'root', 'password')

device.name
'RM5PomeziaSNode'
device.model
'Rocket M5'
device.os
('AirOS', 'XMar7240.v5.3.3.sdk.9634.1111221.2238')

device.to_json()
```

4.3.3 OpenWRT example

Now we try to use OpenWRT instead of AirOS

```
from netengine.backends.ssh import OpenWRT

device = OpenWRT('10.177.8.100', 'root', 'password')

device.name
'owrt1'
device.os
'('OpenWrt', '12.09 (Attitude Adjustment 12.09, r36088)')'

device.to_json()
```

4.4 SNMP backend

4.4.1 SNMP

SNMP (Simple Network Management Protocol) is a network protocol very used to retrieve info from device. All the information are retrieved by using codes called MIBs. All MIBs have a tree-like structure, every main information is the root and by adding more detail to the info the tree gains more depth. Obviously, by getting the smallest MIB which is "1" or simply " . " one can get all the tree.

The SNMP backend provides support for 2 firmwares:

- AirOS
- OpenWRT

4.4.2 AirOS example

```
from netengine.backends.snmp import AirOS
device = AirOS("10.40.0.130")
device.name
'RM5PomeziaSNode'
device.uptime_tuple
(121, 0, 5)  # a tuple containing device uptime hours, mins and seconds
```

We have just called two simple properties on device, but we can ask device for more specific values or portions of the SNMP tree
device.next("1.3.6")

Otherwise, if you want simply a value of the tree just type:: device.get_value("oid_you_want_to_ask_for")

4.4.3 OpenWRT example

The same instructions typed above can be applied to OpenWRT itself, just remember to import the correct firmware by typing:

```
from netengine.backends.snmp import OpenWRT
```

4.5 HTTP backend

This backend is made upon the service provided by AirOS devices, in fact the have an administration page with some dedicated url to take info in json format.

This is what was used to this backend. Since this kind of service is available only for AirOS, this is the only supported firmware.

4.5.1 AirOS example

```
from netengine.backends.snmp import AirOS
device = AirOS("10.40.0.130")
device.name
'RM5PomeziaSNode'
```

Many times you can have to deal with an antenna in mode AP (Access Point) with multiple client attached, in this case the backend gives the list of all connected stations by simply typing:

```
device.connected_stations
```

to have the list of all connected stations with some interesting parameters.

4.6 netengine-utils command line utility

netengine-utils is a shell utility that provides bindings to some features of `netengine.utils`:

- `netengine.utils.ifconfig`
- `netengine.utils.iwfconfig`
- `netengine.utils.manufacturer_lookup`

This utility can be called from the interactive linux shell, bash scripts or from software written in a different language than python.

4.6.1 Convert local ifconfig output to JSON

Convert the **ifconfig** output of the local machine to JSON format:

```
netengine-utils ifconfig
```

Example output:

```
[
    {
        "name": "eth0",
        "link_encap": "Ethernet",
        "hardware_address": "00:26:b9:20:5f:09",
        "inet": "193.206.99.183",
        "broadcast": "193.206.99.255",
        "mask": "255.255.255.128",
        "inet6": "",
        "inet6_local": "fe80::226:b9ff:fe20:5f09/64",
        "mtu": "1500",
        "metric": "1",
```

```
        "rx_packets": "18237538",
        "tx_packets": "12674692",
        "collisions": "0",
        "txqueuelen": "1000",
        "rx_bytes": "2297452870",
        "tx_bytes": "27474972424"
    },
    {
        "name": "lo",
        "link_encap": "Local Loopback",
        "hardware_address": "",
        "inet": "127.0.0.1",
        "broadcast": "",
        "mask": "255.0.0.0",
        "inet6": "::1/128",
        "inet6_local": "",
        "mtu": "65536",
        "metric": "1",
        "rx_packets": "12025",
        "tx_packets": "12025",
        "collisions": "0",
        "txqueuelen": "0",
        "rx_bytes": "2165364",
        "tx_bytes": "2165364"
    },
    {
        "name": "lxcbr0",
        "link_encap": "Ethernet",
        "hardware_address": "c2:b3:22:60:4d:e4",
        "inet": "10.0.3.1",
        "broadcast": "10.0.3.255",
        "mask": "255.255.255.0",
        "inet6": "",
        "inet6_local": "fe80::c0b3:22ff:fe60:4de4/64",
        "mtu": "1500",
        "metric": "1",
        "rx_packets": "0",
        "tx_packets": "229",
        "collisions": "0",
        "txqueuelen": "0",
        "rx_bytes": "0",
        "tx_bytes": "45639"
    }
]
```


4.6.2 Convert specified ifconfig output to JSON

Convert the **ifconfig** output specified with the `--value` option to JSON format:

```
netengine-utils ifconfig --value "$OUTPUT"
```

In which `$OUTPUT` is a variable containing the ifconfig output you obtained from some remote device.

4.6.3 Ifconfig "netjson" option

Pass the `--netjson` option:

```
netengine-utils ifconfig --netjson
```

And you will get a format similar to the following one:

```
[
    {
        "name": "eth0",
        "mac": "00:26:b9:20:5f:09",
        "mtu": "1500",
        "ip": [
            {
                "address": "193.206.99.183/25"
            },
            {
                "address": "fe80::226:b9ff:fe20:5f09/64"
            }
        ]
    },
    {
        "name": "lo",
        "mtu": "65536",
        "ip": [
            {
                "address": "127.0.0.1/8"
            },
            {
                "address": "::1/128"
            }
        ]
    },
    {
        "name": "lxcbr0",
        "mac": "c2:b3:22:60:4d:e4",
        "mtu": "1500",
        "ip": [
            {
                "address": "10.0.3.1/24"
            },
            {
                "address": "fe80::c0b3:22ff:fe60:4de4/64"
            }
        ]
    }
]
```

4.6.4 Convert local iwconfig output to JSON

Convert the **iwconfig** output of the local machine to JSON format:

```
netengine-utils iwconfig
```

Example output:

```
[
    {
        "name": "wlan0",
        "ieee": "802.11abgn",
```

```
        "essid": "off/any",
        "mode": "Managed",
        "access_point": "Not-Associated",
        "tx_power": "off",
        "retry_long_limit": "7",
        "rts_thr": "off",
        "fragment_thr": "off",
        "power_management": "off"
    }
]
```

4.6.5 Convert specified iwconfig output to JSON

Convert the **iwconfig** output specified with the `--value` option to JSON format:

```
netengine-utils iwconfig --value "$OUTPUT"
```

In which `$OUTPUT` is a variable containing the iwconfig output you obtained from some remote device.

4.6.6 Iwconfig "netjson" option

Pass the `--netjson` option:

```
netengine-utils iwconfig --netjson
```

And you will get a format similar to the following one:

```
[
    {
        "name": "wlan0",
        "mac": "DC:9F:DB:27:77:80",
        "wireless": {
            "bitrate": "9 Mb/s",
            "standard": "802.11abgn",
            "essid": "provinciawifi",
            "mode": "sta",
            "rts_threshold": "off",
            "frag_threshold": "off"
        }
    }
]
```

4.6.7 Manufacturer Lookup

Use this command to retrieve the manufacturer from a mac address or prefix:

```
netengine-utils manufacturer_lookup --value 24:a4:3c:00:11:22

Ubiquiti Networks, INC
```

Different formats are allowed:

```
netengine-utils manufacturer_lookup --value 24-A4-3C-00-11-22
netengine-utils manufacturer_lookup --value 24A43C001122
netengine-utils manufacturer_lookup --value 24-A4-3C
```

4.7 Indices and tables

- genindex
- modindex
- search