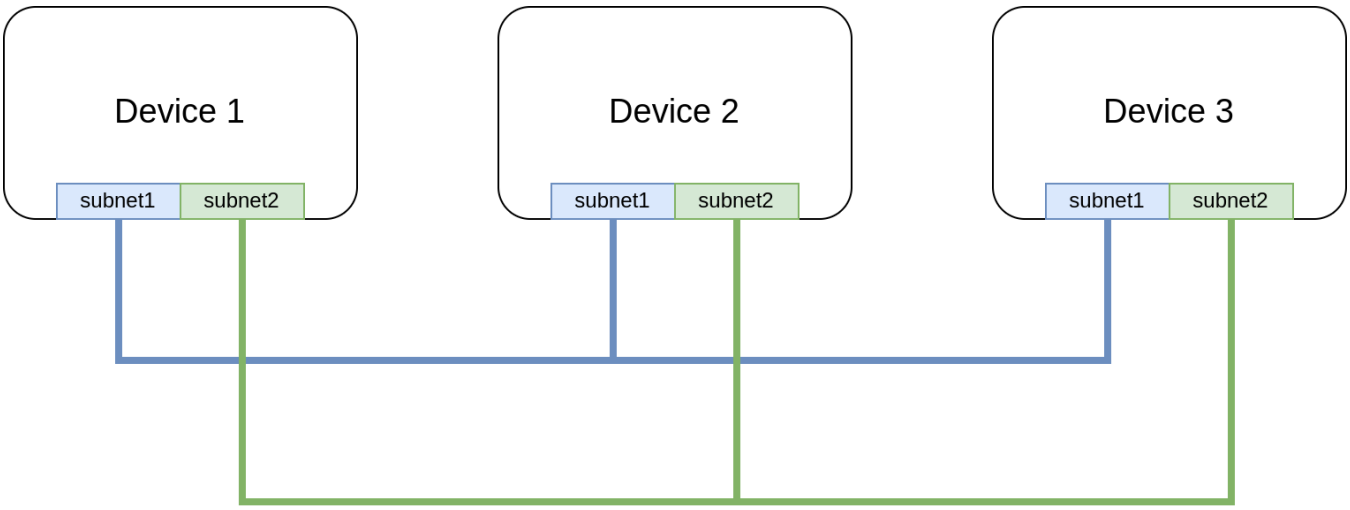
Python Coding Challenge

Project Details

A product your team is designing has multiple devices that communicate with each other using ethernet networks. For redundancy, there are 2 networks for each device. You are responsible for writing a Python library that provides a function which returns a list of IP addresses that are pingable on one network but not the other. We will review your code the same as we would review any pull request at Joby and want to see you give us your best in performance and style! We will set the review up to be anonymous to ensure an unbiased review of your submission.



Explanation

Create a function that takes in 4 arguments: `subnet1` , `subnet2` , `retries` , and `ignore_list` . The `subnet1` and `subnet2` arguments are strings in an IPv4 format that indicate the subnet used with a `/24` `CIDR` notation. For example, a subnet of `192.168.1.0/24` would be a **valid** input. A subnet parameter of `255.255.255.0` or `192.168.1.0` would be an **invalid** input. The retries parameter informs your function of how many attempts it should make when pinging an address and should default to 1 but be no more then 3. The ignore_list parameter is a list of octets that the function should ignore. This function is responsible for pinging all IP addresses on both `subnet1` and `subnet2` and returning a list of IP addresses that are pingable on one subnet but not the other.

`Example 1:`

subnet1	IP addresses for subnet1	pingable	subnet2	IP addresses for subnet2	pingable	Ignored	Ignore List
192.168.1.0/24	192.168.1.24	YES	192.168.2.0/24	192.168.2.24	YES	NO	[]
	192.168.1.25	YES		192.168.2.25	NO	NO	
	192.168.1.26	NO		192.168.2.26	NO	NO	
	192.168.1.27	NO		192.168.2.27	YES	NO	

`In this example, your function must return a list of dictionaries in the following format:`

```
[
    {
        "192.168.1.25": true,
        "192.168.2.25": false
    },
    {
        "192.168.1.27": false,
        "192.168.2.27": true
    }
]
```

We do not care about IP addresses that are pingable on both or IP addresses that are NOT pingable on both. So the following entries would indicate your code is not working as required.

```
[
    {
        "192.168.1.25": true,
        "192.168.2.25": true
    },
     {
        "192.168.1.25": false,
        "192.168.2.25": false
    }
]
```

The `ignore_list` parameter should be a list of integers that represent the last octet of IP addresses to ignore. Your function should log a warning if an octet is provided that is outside the subnet range but should not prevent your function from working correctly. For example, using a subnet of `192.168.1.0/24` , if the octet 280 was provided in the ignore list, a warning should be logged since 280>254. In the following example, these ignore octets are provided in the ignore list.

`Example 2:`

subnet1	ip_address1	pingable	subnet2	ip_address2	pingable	Ignored	Ignore List
192.168.1.0/24	192.168.1.24	YES	192.168.2.0/24	192.168.2.24	YES	NO	`[27]`
	192.168.1.25	YES		192.168.2.25	NO	NO	
	192.168.1.26	NO		192.168.2.26	NO	NO	
	192.168.1.27	NO		192.168.2.27	YES	YES	

`In this example, your function must return the following list of dictionaries.`

```
[
    {
        "192.168.1.25": true,
        "192.168.2.25": false
    }
]
```

Function example

```
subnet1 = '192.168.1.0/24'
subnet2 = '192.168.2.0/24'
retries = 2
ignore_list = [25, 29]

non_pingable_list = your_awesome_function(subnet1, subnet2, retries, ignore_list)
```

Requirements

- Use Python 3.11
- performance matters! Ask yourself, is this an IO bound or CPU bound problem and use multi-threading, multi-processing, or a multi-tasking way of accomplishing the goal. There is a good, better, best approach, so give us the best and be ready to explain the reasoning behind your code!
- A simple `example.py` must be provided to explain how your function is used
- A list of dictionaries must be returned following the above format.
- use logging library for outputting to console, do not use print
- Package your code with a `setup.py` . We must be able to install your code with pip.
- Feel free to use any third party library, but any package used outside of the Python standard library should be declared as a dependency in a `requirements.txt` or a `setup.py` .
- Unit test should be capable of running on their own, completely independent of their environment. In other words, your unit test should not require an internet connection and should not test functions outside the scope of your code.
- General guidelines to follow for unit testing:
    - test small segments of code
    - Utilize mocking to ensure only your code is executed
    - follow test naming convention, ie. all functions start with "test" and all test modules start with "test"
    - break your main code up into small testable functions that make unit testing easier
- Package should be submitted as a zip file or tar.

Style

- Follow PEP8 standards. Your code will be ran against a linter (Pylint, Flake8) so make sure it passes linting
- Your function can belong to a Python module or a Python class, but please do not use just one function! break it up and follow clean coding principles that follow the Zen of Python
- Use docstrings for ALL functions and module level docstrings. Pylint and Flake8 should give you an error if your docstrings are incorrect
- Inline comments when necessary to explain a difficult piece of code, but most code should be self explanatory and function names should be sufficient
- No magic numbers should be used.
- Show us you know Python. Remember, python is not C++. It has a rich set of built in features that are optimized for performance.
- Type hinting is nice but not required if docstring indicates types.