fauxmo Documentation

Release 0.4.1

Nathan Henrie

Apr 08, 2017

Contents

Contents:

CHAPTER 1

fauxmo package

Subpackages

fauxmo.plugins package

Submodules

fauxmo.plugins.simplehttpplugin module

simplehttpplugin.py

Fauxmo plugin that makes simple HTTP requests in its *on* and *off* methods. Comes pre-installed in Fauxmo as an example for user plugins.

For more complicated requests (e.g. authentication, sending JSON), check out RESTAPIPlugin in *https://github.com/n8henrie/fauxmo-plugins/*, which takes advantage of Requests' rich API.

class `fauxmo.plugins.simplehttpplugin.`**`SimpleHTTPPlugin`**(**, name: str, port: int, on_cmd: str, off_cmd: str, method: str = 'GET', on_data: dict = None, off_data: dict = None, headers: dict = None, user: str = None, password: str = None*) → None

Bases: *fauxmo.plugins.FauxmoPlugin*

Plugin for interacting with HTTP devices.

The Fauxmo class expects plguins to be instances of objects that inherit from FauxmoPlugin and have on() and off() methods that return True on success and False otherwise. This class takes a mix of url, method, header, body, and auth data and makes REST calls to a device.

`__init__`(**, name: str, port: int, on_cmd: str, off_cmd: str, method: str = 'GET', on_data: dict = None, off_data: dict = None, headers: dict = None, user: str = None, password: str = None*) → None

Initialize a SimpleHTTPPlugin instance

Keyword Arguments

- **on_cmd** – URL to be called when turning device on
- **off_cmd** – URL to be called when turning device off
- **method** – HTTP method to be used for both *on()* and *off()*
- **on_data** – HTTP body to turn device on
- **off_data** – HTTP body to turn device off
- **headers** – Additional headers for both *on()* and *off()*
- **user** – Username for HTTP authentication (basic or digest only)
- **password** – Password for HTTP authentication (basic or digest only)

off() → bool

on() → bool

set_state(*cmd: str*, *data: str*) → bool
Call HTTP method, for use by *functools.partialmethod*.

Parameters

- **cmd** – Either *"on_cmd"* or *"off_cmd"*, for *getattr(self, cmd)*
- **data** – Either *"on_data"* or *"off_data"*, for *getattr(self, data)*

Returns Boolean indicating whether it state was set successfully

Module contents

fauxmo.plugins :: Provide ABC for Fauxmo plugins.

class fauxmo.plugins.**FauxmoPlugin**(*, *name: str*, *port: int*) → None
Bases: abc.ABC

ABC for Fauxmo plugins

This will become the *plugin* attribute of a *Fauxmo* instance. Its *on* and *off* methods will be called when Alexa turns something *on* or *off*.

All keys (other than the list of *DEVICES*) from the config will be passed into FauxmoPlugin as kwargs at initialization, which should let users do some interesting things. However, that means users employing custom config keys will need to override *__init__* and either set the *name* and "private" *_port* attributes manually or pass the appropriate args to *super().__init__()*.

__init__(*, *name: str*, *port: int*) → None
Initialization for FauxmoPlugin

Keyword Arguments

- **name** – Required, device name
- **port** – Required, port that the Fauxmo associated with this plugin should run on

Note about *port*: if not given in config, it will be set to an apparently free port in *fauxmo.fauxmo* before FauxmoPlugin initialization. This attribute serves no default purpose in the FauxmoPlugin but is passed in to be accessible by user code (i.e. for logging / debugging). Alternatively, one could accept and throw away the passed in *port* value and generate their own port in a plugin, since the Fauxmo device determines its port from the plugin's instance attribute.

name

off() → bool
Called when Alexa turns this Fauxmo device off

on() → bool
Called when Alexa turns this Fauxmo device on

port

Submodules

fauxmo.cli module

fauxmo.fauxmo module

fauxmo.protocols module

fauxmo.utils module

Module contents

CHAPTER 2

Credits

Development Lead

- Maker Musings https://github.com/makermusings

Contributors

- Modifications by Nathan Henrie nate@n8henrie.com

CHAPTER 3

Changelog

Will not contain minor changes – feel free to look through `git log` for more detail.

v0.4.0 :: 20170402

- Rename handlers to plugins
- Add interface for user plugins
- Add type hints
- Require Python 3.6
- Eliminate third party dependencies
- Make sure to close connection when plugin commands fail / return False

v0.3.3 :: 20160722

- Added compatibility for `rollershutter` to `handlers.hass`
- Changed `handlers.hass` to send values from a dict to make addition of new services easier in the future

v0.3.2 :: 20160419

- Update SSDPServer to `setsockopt` to permit receiving multicast broadcasts
- `sock` kwarg to `create_datagram_endpoint` no longer necessary, restoring functionality to Python 3.4.0 - 3.4.3 (closes #6)
- `make_udp_sock()` no longer necessary, removed from `fauxmo.utils`

- Tox and Travis configs switched to use Python 3.4.2 instead of 3.4.4 (since 3.4.2 is the latest available in the default Raspbian Jessie repos)

v0.3.1 :: 20160415

- Don't decode the UDP multicast broadcasts (hopefully fixes #7)
  - They might not be from the Echo and might cause a `UnicodeDecodeError`
  - Just search the bytes instead
- Tests updated for this minor change

v0.3.0 :: 20160409

- Fauxmo now uses asyncio and requires Python >= 3.4.4
- *Extensive* changes to codebase
- Handler classes renamed for PEP8 (capitalization)
- Moved some general purpose functions to `fauxmo.utils` module
- Both the UDP and TCP servers are now in `fauxmo.protocols`
- Added some rudimentary pytest tests including tox and Travis support
- Updated documentation on several classes

v0.2.0 :: 20160324

- Add additional HTTP verbs and options to `RestApiHandler` and Indigo sample to config
  - **NB:** Breaking change: `json` config variable now needs to be either `on_json` or `off_json`
- Make `RestApiHandler` DRYer with `functools.partialmethod`
- Add `SO_REUSEPORT` to `upnp.py` to make life easier on OS X

v0.1.11 :: 20160129

- Consolidate logger to `__init__.py` and import from there in other modules

v0.1.8 :: 20160129

- Add the ability to manually specify the host IP address for cases when the auto detection isn't working (https://github.com/n8henrie/fauxmo/issues/1)
- Deprecated the `DEBUG` setting in `config.json`. Just use `-vvv` from now on.

v0.1.6 :: 20160105

- Fix for Linux not returning local IP
  - restored method I had removed from Maker Musings original / pre-fork version not knowing it would introduce a bug where Linux returned 127.0.1.1 as local IP address

v0.1.4 :: 20150104

- Fix default verbosity bug introduced in 1.1.3

v0.1.0 :: 20151231

- Continue to convert to python3 code
- Pulled in a few PRs by @DoWhileGeek working towards python3 compatibility and improved devices naming with dictionary
- Renamed a fair number of classes
- Added kwargs to several class and function calls for clarity
- Renamed several variables for clarity
- Got rid of a few empty methods
- Import devices from `config.json` and include a sample
- Support `POST`, headers, and json data in the RestApiHandler
- Change old debug function to use logging module
- Got rid of some unused dependencies
- Moved license (MIT) info to LICENSE
- Added argparse for future console scripts entry point
- Added Home Assistant API handler class
- Use "string".format() instead of percent
- Lots of other minor refactoring

CHAPTER 4

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

Types of Contributions

Report Bugs

Report bugs at https://github.com/n8henrie/fauxmo/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

Write Documentation

fauxmo could always use more documentation, whether as part of the official fauxmo docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/n8henrie/fauxmo/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

Get Started!

Ready to contribute? Here's how to set up fauxmo for local development.

1. Fork the fauxmo repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/fauxmo.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ cd fauxmo
$ python3 -m venv venv
$ source venv/bin/activate
$ pip install -e .
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 fauxmo tests
$ python3 setup.py test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests if I am using tests in the repo.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.md
3. The pull request should work for Python 3.6. If I have included a `.travis.yml` file in the repo, check https://travis-ci.org/n8henrie/fauxmo/pull_requests and make sure that the tests pass for all supported Python versions.

Tips

To run a subset of tests: `py.test tests/test_your_test.py`

CHAPTER 5

Fauxmo README

Python 3 module that emulates Belkin WeMo devices for use with the Amazon Echo.

Originally forked from https://github.com/makermusings/fauxmo, unforked to enable GitHub code search (which currently doesn't work in a fork), and because the libraries have diverged substantially.

- Documentation: fauxmo.readthedocs.org

Introduction

The Amazon Echo is able to control certain types of home automation devices by voice. Fauxmo provides emulated Belkin Wemo devices that the Echo can turn on and off by voice, locally, and with minimal lag time. Currently these Fauxmo devices can be configured to make requests to an HTTP server or to a Home Assistant instance via its Python API and only require a JSON config file for setup.

As of version v0.4.0, Fauxmo uses several API features and f-strings that require Python 3.6+. I highly recommend looking into pyenv if you're currently on an older Python version and willing to upgrade. Otherwise, check out the FAQ section at the bottom for tips on installing an older Fauxmo version (though note that I will not be continuing development or support for older versions).

Terminology

faux (`\fō\`): imitation WeMo: Belkin home automation product with which the Amazon Echo can interface

Fauxmo (`\fō-mō\`): Python 3 module that emulates Belkin WeMo devices for use with the Amazon Echo.

Fauxmo has a server component that helps register "devices" with the Echo (which may be referred to as the Fauxmo server or Fauxmo core). These devices are then exposed individually, each requiring its own port, and may be referred to as a Fauxmo device or a Fauxmo instance. The Echo interacts with each Fauxmo device as if it were a separate WeMo device.

Usage

Installation into a venv is *highly recommended*, especially since it's baked into the recent Python versions that Fauxmo requires.

Simple install: From PyPI

1. `python3 -m venv venv`
2. `source venv/bin/activate`
3. `python3 -m pip install fauxmo`
4. Make a `config.json` based on `config-sample.json`
5. `fauxmo -c config.json [-v]`

Simple install of dev branch from GitHub (for testing features in

development – for actually contributing to development clone the repo as per below)

1. `python3 -m venv venv`
2. `source venv/bin/activate`
3. `pip install [-e] git+https://github.com/n8henrie/fauxmo.git@dev`

Install for development from GitHub

1. `git clone https://github.com/n8henrie/fauxmo.git`
2. `cd fauxmo`
3. `python3 -m venv venv`
4. `source venv/bin/activate`
5. `pip install -e .[dev]`
6. `cp config-sample.json config.json`
7. Edit `config.json`
8. `fauxmo [-v]`

Set up the Echo

1. Open the Amazon Alexa webapp to the Smart Home page
2. **With Fauxmo running**, click "Discover devices" (or tell Alexa to "find connected devices")
3. Ensure that your Fauxmo devices were discovered and appear with their names in the web interface
4. Test: "Alexa, turn on [the kitchen light]"

Set fauxmo to run automatically in the background

NB: As discussed in #20, the example files in `extras/` are *not* included when you install from PyPI* (using `pip`). If you want to use them, you either need to clone the repo or you can download them individually using tools like `wget` or `curl` by navigating to the file in your web browser, clicking the `Raw` button, and using the resulting URL in your address bar.

* As of Fauxmo v0.4.0 `extras/` has been added to `MANIFEST.in` and may be included somewhere depending on installation from the `.tar.gz` vs `whl` format – if you can't find them, you should probably just get the files manually as described above.

systemd (e.g. Raspbian Jessie)

1. Recommended: add an unprivileged user to run Fauxmo: `sudo useradd -r -s /bin/false fauxmo`
    - NB: Fauxmo may require root privileges if you're using ports below 1024
2. `sudo cp extras/fauxmo.service /etc/systemd/system/fauxmo.service`
3. Edit the paths in `/etc/systemd/system/fauxmo.service`
4. `sudo systemctl enable fauxmo.service`
5. `sudo systemctl start fauxmo.service`

launchd (OS X)

1. `cp extras/com.n8henrie.fauxmo.plist ~/Library/LaunchAgents/com.n8henrie.fauxmo.plist`
2. Edit the paths in `~/Library/LaunchAgents/com.n8henrie.fauxmo.plist`
    - You can remove the `StandardOutPath` and `StandardErrorPath` sections if desired
3. `launchctl load ~/Library/LaunchAgents/com.n8henrie.fauxmo.plist`
4. `launchctl start com.n8henrie.fauxmo`

Plugins

Plugins are small user-extendible classes that allow users to easily make their own actions for Fauxmo to run by way of Alexa commands. They were previously called Handlers and may be referred to as such in places in the code and documentation.

Fauxmo v0.4.0 implements a new and breaking change in the way Handlers were implemented in previous versions, which requires modification of the `config.json` file (as described below).

A few plugins and the ABC from which the plugins are required to inherit are included and installed by default in the `fauxmo.plugins` package. The pre-installed plugins, like the rest of the core Fauxmo code, have no third party dependencies.

The pre-installed plugins include

- `fauxmo.plugins.simplehttpplugin.SimpleHTTPPlugin`
- `fauxmo.plugins.command_line.CommandLinePlugin`

`SimpleHTTPPlugin` responds to Alexa's `on` and `off` commands by making requests to URL endpoints by way of `urllib`. Example uses cases relevant to the IOT community might be a Flask server served from localhost that provides a nice web interface for toggling switches, whose endpoints could be added as the `on_cmd` and `off_cmd` args to a `SimpleHTTPPlugin` instance to allow activation by way of Alexa -> Fauxmo.

Please see details regarding `SimpleHTTPPlugin` configuration in the class's docstring, which I intend to continue as a convention for Fauxmo plugins. Users hoping to make more complicated requests may be interested in looking at `RESTAPIPlugin` in the `fauxmo-plugins repository`, which uses Requests for a much friendlier API.

User plugins

Users can easily create their own plugins, which is the motivation behind most of the changes in Fauxmo v0.4.0.

To get started:

1. Decide on a name for your plugin class. I highly recommend something descriptive, CamelCase and a `Plugin` suffix, e.g. `FooSwitcherPlugin`.
2. I strongly recommend naming your module the same as the plugin, but in all lower case, e.g. `fooswitcherplugin.py`.
3. Note the path to your plugin, which will need to be included in your `config.json` as `path` (absolute path recommended, ~ for homedir is okay).
4. Write your class, which should at minimum:
   - inherit from `fauxmo.plugins.FauxmoPlugin`.
   - provide the methods `on()` and `off()`.
5. Any required settings will be read from your `config.json` and passed into your plugin as kwargs at initialization, see below.

In addition to the above, if you intend to share your plugin with others, I strongly recommend that you:

- Include generous documentation as a module level docstring.
- Note specific versions of any dependencies in that docstring.
  - Because these user plugins are kind of "side-loaded," you will need to manually install their dependencies into the appropriate environment, so it's important to let other users know exactly what versions you use.

Notable plugin examples

NB: You may need to *manually* install additional dependencies for these to work – look for the dependencies in the module level docstring.

- https://github.com/n8henrie/fauxmo-plugins
  - `RESTAPIPlugin`
    - Trigger HTTP requests with your Echo.
    - Similar to `SimpleHTTPPlugin`, but uses Requests for a simpler API and easier modification.
  - `HassAPIPlugin`
    - Uses the Home Assistant Python API to run commands through a local or remote Home Assistance instance.
  - `CommandLinePlugin`
    - Run a shell command on the local machine.

  - User contributions of interesting plugins are more than welcome!

Configuration

I recommend that you copy and modify `config-sample.json`. Fauxmo will use whatever config file you specify with `-c` or will search for `config.json` in the current directory, `~/.fauxmo/`, and `/etc/fauxmo/` (in that order). The minimal configuration settings are:

- `FAUXMO`: General Fauxmo settings
  - `ip_address`: Optional[str] - Manually set the server's IP address. Recommended value: `"auto"`.
- `PLUGINS`: Top level key for your plugins, values should be a dictionary of (likely CamelCase) class names, spelled identically to the plugin class, with each plugin's settings as a subdictionary.
  - `ExamplePlugin`: Your plugin class name here, case sensitive.
    * `path`: The absolute path to the Python file in which the plugin class is defined (please see the section on user plugins above). Required for user plugins / plugins not pre-installed in the `fauxmo.plugins` subpackage.
    * `example_var1`: For convenience and to avoid redundancy, your plugin class can *optionally* use config variables at this level that will be shared for all `DEVICES` listed in the next section (e.g. an api key that would be shared for all devices of this plugin type). If provided, your plugin class must consume this variable in a custom `__init__`.
    * `DEVICES`: List of devices that will employ `ExamplePlugin`
      · `name`: Optional[str] – Name for this device. Optional in the sense that you can leave it out of the config as long as you set it in your plugin code as the `_name` attribute, but it does need to be set somewhere. If you omit it from config you will also need to override the `__init__` method, which expects a `name` kwarg.
      · `port`: Optional[int] – Port that Echo will use connect to device. Should be different for each device, Fauxmo will attempt to set automatically if absent from config. NB: Like `name`, you can choose to set manually in your plugin code by overriding the `_port` attribute (and the `__init__` method, which expects a `port` kwarg otherwise).
      · `example_var2`: Config variables for individual Fauxmo devices can go here if needed (e.g. the URL that should be triggered when a device is activated). Again, your plugin class will need to consume them in a custom `__init__`.

Each user plugin should describe its required configuration in its module-level docstring. The only required config variables for all plugins is `DEVICES`, which is a `List[dict]` of configuration variables for each device of that plugin type. Under `DEVICES` it is a good idea to set a fixed, high, free `port` for each device, but if you don't set one, Fauxmo will try to pick a reasonable port automatically (though it will change for each run).

Please see `config-sample` for a more concrete idea of the structure of the config file, using the built-in `SimpleHTTPPlugin` for demonstration purposes. Below is a description of the kwargs that `SimpleHTTPPlugin` accepts.

- `name`: What you want to call the device (how to activate by Echo)
- `port`: Port the Fauxmo device will run on
- `on_cmd`: str – URL that should be requested to turn device on.
- `off_cmd`: str – URL that should be requested to turn device off.
- `method`: Optional[str] = GET – GET, POST, PUT, etc.

- `headers`: Optional[dict] – Extra headers
- `on_data` / `off_data`: Optional[dict] – POST data
- `user` / `password`: Optional[str] – Enables HTTP authentication (basic or digest only)

Security considerations

Because Fauxmo v0.4.0+ loads any user plugin specified in their config, it will run untested and potentially unsafe code. If an intruder were to have write access to your `config.json`, they could cause you all kinds of trouble. Then again, if they already have write access to your computer, you probably have bigger problems. Consider making your config.json `0600` for your user, or perhaps `0644 root:YourFauxmoUser`. Use Fauxmo at your own risk, with or without user plugins.

Troubleshooting / FAQ

- How can I increase my logging verbosity?
  - `-v[vv]`
- How can I ensure my config is valid JSON?
  - `python -m json.tool < config.json`
  - Use `jsonlint` or one of numerous online tools
- How can I install an older / specific version of Fauxmo?
  - Install from a tag:
    * `pip install git+git://github.com/n8henrie/fauxmo.git@v0.1.11`
  - Install from a specific commit:
    * `pip install git+git://github.com/n8henrie/fauxmo.git@d877c513ad45cbbbd77b1b83e7a2f03bf0004856`
- Where can I get more information on how the Echo interacts with devices like Fauxmo?
  - Check out `protocol_notes.md`

Installing Python 3.6 with pyenv

```
sudo install -o $(whoami) -g $(whoami) -d /opt/pyenv
git clone https://github.com/yyuu/pyenv /opt/pyenv
echo 'export PYENV_ROOT="/opt/pyenv"' >> ~/.bashrc
echo 'export PATH="$PYENV_ROOT/bin:$PATH"' >> ~/.bashrc
echo 'eval "$(pyenv init -)"' >> ~/.bashrc
source ~/.bashrc
pyenv install 3.6.1
```

You can then install Fauxmo into Python 3.6 in a few ways, including:

```
# Install with pip
"$(pyenv root)"/versions/3.6.1/bin/python3.6 -m pip install fauxmo

# Show full path to fauxmo console script
```

```
pyenv which fauxmo

# Run with included console script
fauxmo -c /path/to/config.json -vvv

# I recommend using the full path for use in start scripts (e.g. systemd, cron)
"$(pyenv root)"/versions/3.6.1/bin/fauxmo -c /path/to/config.json -vvv

# Alternatively, this also works (after `pip install`)
"$(pyenv root)"/versions/3.6.1/bin/python3.6 -m fauxmo.cli -c config.json -vvv
```

Acknowledgements / Reading List

- Tremendous thanks to @makermusings for the original version of Fauxmo!
  - Also thanks to @DoWhileGeek for commits towards Python 3 compatibility
- http://www.makermusings.com/2015/07/13/amazon-echo-and-home-automation
- http://www.makermusings.com/2015/07/18/virtual-wemo-code-for-amazon-echo
- http://hackaday.com/2015/07/16/how-to-make-amazon-echo-control-fake-wemo-devices
- https://developer.amazon.com/appsandservices/solutions/alexa/alexa-skills-kit
- https://en.wikipedia.org/wiki/Universal_Plug_and_Play
- http://www.makermusings.com/2015/07/19/home-automation-with-amazon-echo-apps-part-1
- http://www.makermusings.com/2015/08/22/home-automation-with-amazon-echo-apps-part-2

CHAPTER 6

protocol_notes.md

Details on the Echo's interaction with Fauxmo, and how to examine it for debugging.

Tons of information gathered by @makermusings, I *strongly* recommend you start by reading these:

- https://github.com/makermusings/fauxmo/blob/master/protocol_notes.txt
- http://www.makermusings.com/2015/07/13/amazon-echo-and-home-automation

In summary:

1. User tells Echo to "find connected devices" or clicks corresponding button in webapp
2. Echo broadcasts "device search" to `239.255.255.250:1900` (UDP)
3. Fauxmo response includes `LOCATION` of `setup.xml` endpoint for each "device" in config (UDP)
4. Echo requests `setup.xml` endpoint at above `LOCATION` (HTTP) for each device
5. Fauxmo responds with setup information for each device (HTTP)
6. Alexa verbally announces any discovered devices (*really* wish I could mute this – set volume to 1 beforehand if I'll be doing it a bunch), and they also show up in the webapp

Once you understand the basic model of interaction, the next step in debugging is to inspect the actual requests and responses.

The following commands require some tools you might not have by default; you can get them with: `sudo apt-get install tcpdump tshark nmap`. Doesn't matter what you choose regarding the wireshark question you'll get during installation; just read the warning and make a good decision. On OSX, use homebrew to install the same.

First, get the IP address of your Echo. If you don't know it:

```
# Assuming your local subnet is 192.168.27.*
sudo nmap -sP 192.168.27.1/24 | grep -i -B 2 F0:27:2D
```

You should get `Nmap scan report for 192.168.27.XXX` – your Echo IP address. For the examples below, I'll use `192.168.27.100` as the Echo IP address, and `192.168.27.31` as the Pi's IP address (31 as in 3.14, easier to remember).

Next, we'll check out the info being sent to and from the Echo and Fauxmo. In one window, run Fauxmo in verbose mode. In a second window, run the commands below, and check their output when you tell the Echo to find connected devices.

To get an overview of what's going on, start with `tshark`:

```
sudo tshark -f "host 192.168.27.100" -Y "udp"

# Alternatively, only show the Echo's SEARCH requests:
sudo tshark -f "host 192.168.27.100" -Y "udp contains SEARCH"

# Only show the Fauxmo responses (note still using the Echo's IP):
sudo tshark -f "host 192.168.27.100" -Y "udp contains LOCATION"
```

Example output for the first command, showing a few sets of SSDP SEARCH sent by the Echo followed by 4 responses by Fauxmo (1 for each device in sample config).

```
Capturing on 'eth0'
  1   0.000000 192.168.27.100 -> 239.255.255.250 SSDP 149 M-SEARCH * HTTP/1.1
  2   0.046414 192.168.27.31 -> 192.168.27.100 SSDP 428 HTTP/1.1 200 OK
  3   0.064351 192.168.27.31 -> 192.168.27.100 SSDP 428 HTTP/1.1 200 OK
  4   0.082011 192.168.27.31 -> 192.168.27.100 SSDP 428 HTTP/1.1 200 OK
  5   0.101093 192.168.27.31 -> 192.168.27.100 SSDP 428 HTTP/1.1 200 OK
  6   0.104016 192.168.27.100 -> 239.255.255.250 SSDP 149 M-SEARCH * HTTP/1.1
  7   0.151414 192.168.27.31 -> 192.168.27.100 SSDP 428 HTTP/1.1 200 OK
  8   0.171049 192.168.27.31 -> 192.168.27.100 SSDP 428 HTTP/1.1 200 OK
  9   0.191602 192.168.27.31 -> 192.168.27.100 SSDP 428 HTTP/1.1 200 OK
 10   0.199882 192.168.27.31 -> 192.168.27.100 SSDP 428 HTTP/1.1 200 OK
 11   0.231841 192.168.27.100 -> 239.255.255.250 SSDP 164 M-SEARCH * HTTP/1.1
 12   0.333406 192.168.27.100 -> 239.255.255.250 SSDP 164 M-SEARCH * HTTP/1.1
```

To get a raw look at all the info, use `tcpdump`. I've cleaned up a bunch of garbage in the below output, but you should still be able to recognize each of the critical components.

```
sudo tcpdump -s 0 -i eth0 -A host 192.168.27.100
```

This should show a ton of detailed info, including all responses sent to / from the Echo. Replace `etho` with your network interface (check with `ip link`) and `192.168.27.100` with your Echo's IP address.

The output should start with several of the Echo's UDP based discovery requests, where you can recognize the `UDP` protocol being sent from the Echo `192.168.27.100` to the network's multicast broadcast `239.255.255.250.1900`, something like:

```
15:48:39.268125 IP 192.168.27.100.50000 > 239.255.255.250.1900: UDP, length 122
M-SEARCH * HTTP/1.1
HOST: 239.255.255.250:1900
MAN: "ssdp:discover"
MX: 15
ST: urn:Belkin:device:**
```

Below that, you should see Fauxmo's responses, also UDP, one for each device in the config. This response provides the Echo with the `LOCATION` of the device's `setup.xml`.

```
15:48:39.513741 IP 192.168.27.31.1900 > 192.168.27.100.50000: UDP, length 386
HTTP/1.1 200 OK
CACHE-CONTROL: max-age=86400
DATE: Sun, 24 Apr 2016 21:48:39 GMT
EXT:
```

```
LOCATION: http://192.168.27.31:12340/setup.xml
OPT: "http://schemas.upnp.org/upnp/1/0/"; ns=01
01-NLS: c66d1ad0-707e-495e-a21a-1d640eed4547
SERVER: Unspecified, UPnP/1.0, Unspecified
ST: urn:Belkin:device:**
USN: uuid:Socket-1_0-2d4ac336-8683-3660-992a-d056b5382a8d::urn:Belkin:device:**
```

Somewhere below that, you'll see the Echo request each device's `setup.xml` (based on the `LOCATION` from the prior step), this time TCP instead of UDP.

```
15:48:39.761878 IP 192.168.27.100.39720 > 192.168.27.31.12341: Flags [P.], seq 1:68,␣
↪ack 1, win 274, options [nop,nop,TS val 619246756 ecr 140303456], length 67
GET /setup.xml HTTP/1.1
Host: 192.168.27.31:12341
Accept: */*
```

And somewhere below that, Fauxmo's setup response, for each device in the config, also TCP:

```
15:48:39.808164 IP 192.168.27.31.12342 > 192.168.27.100.59999: Flags [P.], seq 1:608,␣
↪ack 68, win 453, options [nop,nop,TS val 140303462 ecr 619246754], length 607
HTTP/1.1 200 OK
CONTENT-LENGTH: 375
CONTENT-TYPE: text/xml
DATE: Sun, 24 Apr 2016 21:48:39 GMT
LAST-MODIFIED: Sat, 01 Jan 2000 00:01:15 GMT
SERVER: Unspecified, UPnP/1.0, Unspecified
X-User-Agent: Fauxmo
CONNECTION: close

<?xml version="1.0"?>
<root>
<device>
<deviceType>urn:Fauxmo:device:controllee:1</deviceType>
<friendlyName>fake hass switch by REST API</friendlyName>
<manufacturer>Belkin International Inc.</manufacturer>
<modelName>Emulated Socket</modelName>
<modelNumber>3.1415</modelNumber>
<UDN>uuid:Socket-1_0-cbc4bc63-e0e2-3a78-8a9f-f0ff7e419b79</UDN>
</device>
</root>
```

Then, to get a *really* close look at a request, well go back to `tshark`. For example, we can add the `-V` flag to get a **ton** more info, and add `-c 1` (count) to limit to capturing a single packet, and further refine the capture filter by specifying that we only want to look at packets sent **from** the Pi **to** the Echo.

```
sudo tshark -f "src 192.168.27.31 and dst 192.168.27.100" -c 1 -V
```

At the bottom, you should find the `Hypertext Transfer Protocol` section contains the same `setup.xml` response we found in the `tcpdump` output above.

You can also send requests from another device on the network to check out Fauxmo's responses and ensure that they're getting through the network. For example, to simulate the Echo's device search, run the following from another device on the network, in two different windows:

```
# Seems to work with `nc.traditional` on Raspberry Pi, not yet working for me on OSX
# Window 1: Listen for response on port 12345 (should show up once second command is␣
↪sent)
nc.traditional -l -u -p 12345
```

```
# Window 2: Send simulated UDP broadcast device search (from port 12345)
echo -e '"ssdp:discover"urn:Belkin:device:**' | nc.traditional -b -u -p 12345 239.255.
↪255.250 1900
```

To request a device's `setup.xml`, using the device's `port` from `config.json`:

```
# Send a request for the `setup.xml` of a device from the sample config
curl -v 192.168.27.31:12340/setup.xml
```

The above commands may seem a little complicated if you're unfamiliar, but they're immensely powerful and indispensable for debugging these tricky network issues. If you're not already familiar with them, learning the basics will serve you well in your IoT endeavors!

To verify that Fauxmo is working properly, check for a few things:

1. Is the Pi consistently seeing the Echo's `M-SEARCH` requests?
2. Is Fauxmo consistently replying with the `LOCATION` responses?
3. Is the Echo then requesting the `setup.xml` (for each device)?
4. Is Fauxmo consistently replying with the setup info?

If you can confirm that things seem to be working through number 4, then it would seem that Fauxmo is working properly, and the issue would seem to be elsewhere.

On and Off Commands

One way to examine exactly what the Echo sends to one of your connected Fauxmo devices (i.e. one that *already* works as expected) is to first **stop** Fauxmo (to free up the port), then use netcat to listen to that port while you trigger the command. E.g. for a Fauxmo device configured to use port `12345`, run `nc.traditional -l 12345` and then tell the Echo to "turn on [device name]". The Echo will notify you that the command failed, obviously, because Fauxmo isn't running, but you should be able to see exactly what the Echo sent.

These are the requests that the Echo sends to Fauxmo when you ask it to turn a device...

On

```
POST /upnp/control/basicevent1 HTTP/1.1
Host: 192.168.27.31:12345
Accept: */*
Content-type: text/xml; charset="utf-8"
SOAPACTION: "urn:Belkin:service:basicevent:1#SetBinaryState"
Content-Length: 299

<?xml version="1.0" encoding="utf-8"?>
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/" s:encodingStyle=
↪"http://schemas.xmlsoap.org/soap/encoding/">
<s:Body>
<u:SetBinaryState xmlns:u="urn:Belkin:service:basicevent:1">
<BinaryState>1</BinaryState>
</u:SetBinaryState>
</s:Body>
</s:Envelope>
```

Off

```
POST /upnp/control/basicevent1 HTTP/1.1
Host: 192.168.27.31:12345
Accept: */*
Content-type: text/xml; charset="utf-8"
SOAPACTION: "urn:Belkin:service:basicevent:1#SetBinaryState"
Content-Length: 299

<?xml version="1.0" encoding="utf-8"?>
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/" s:encodingStyle=
↪"http://schemas.xmlsoap.org/soap/encoding/">
<s:Body>
<u:SetBinaryState xmlns:u="urn:Belkin:service:basicevent:1">
<BinaryState>0</BinaryState>
</u:SetBinaryState>
</s:Body>
</s:Envelope>
```

Several similar terms can be used instead of `On` and `Off`, e.g. `Open` and `Close`; the response looks identical. This Reddit post has a good number more that work. NB: the `Dim` commands in the post don't seem to work (likely incompatible with Wemo devices, so the Echo doesn't even try to send them).

CHAPTER 7

Indices and tables

- genindex
- modindex

Python Module Index

f

Index

Symbols

F

N

O

P

S