doe-nf Documentation

Release 1.0

UH Networking Lab

Nov 12, 2018

Contents

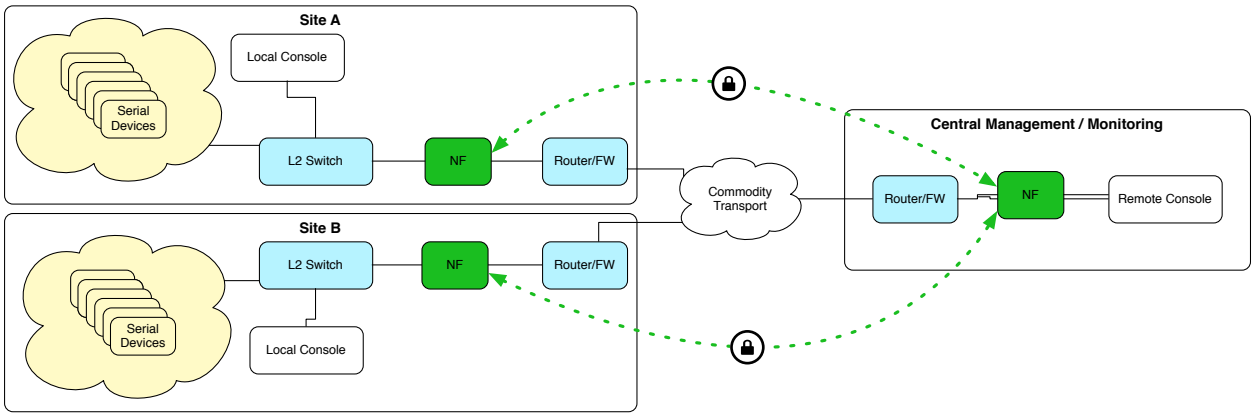
CHAPTER 1

Overview and Quickstart

1.1 Project Overview

The DOE Network Function (NF) Project is designed to provide functionality to Industrial Control System (ICS) operators to allow them to set per-flow policy between endpoints. Primarily this is intended to provide additional security and behavior stability over insecure and unreliable commodity transport, but the NF can be leveraged for a wide range of needs as they evolve over time. By using a network function to provide additional functionality the lifecycles of network resilience needs and industrial hardware are decoupled, allowing long-life ICS hardware to continue to operate in a fast evolving network environment.

The Network Function insertion in the topology is generally as seen below:



While this trivial example shows paired NFs being used to provide secure and reliable delivery over commodity transport, it can also be deployed to enforce flow policy within an administrative domain - for example, between the typical enterprise network and the industrial controls at the same site.

1.2 Reference Function and Dependencies

The reference Network Function here is not intended to be performant, but rather a proof of concept for evaluating basic behavior in the face of differing network conditions and flow inputs. The ultimate output of the project is a specification that we expect ICS vendors will use to implement the functionality in their own products.

The reference NF, test orchestration framework, and analysis tools are written in Python. Below we list both the software and other requirements for utilizing these tools.

1.2.1 Test Topology Builder (`reserve.py`)

To build the test topology using the `reserve.py` script you will need a valid GENI credential in order to use the VTS testbed employed. If you do not have GENI credentials you can find instructions for acquiring credentials at the NSF GENI Portal.

Your environment will require the installation of the following python libraries:

- geni-lib (properly configured with your GENI credentials)

Note: The default Vagrant installation of `geni-lib` builds a 32-bit virtual machine whose base OS has conflicts with the libraries required for operation of this experiment. You can use `virtualenv` to build an isolated environment on that VM to separate your use of this code from the base OS.

- uhgeni
- requests
- lxml
- ipaddress
- cryptography

The topology builder will function with the libraries above, although it sets up an environment that will require that you link a Dropbox account with the VTS site you use, in order to automatically transfer the experiment data out of the isolated topology that VTS creates. You can easily do this using the documentation here.

1.2.2 Experiment Runner (`runexp.py`)

The experiment orchestration system does not require that you use a topology in the GENI environment, but the experiment runner script provided here has weak dependencies on that topology. If you have an interest in setting up the experiment topology on other resources and still using the `runexp.py` script in your environment, please contact UH Netlab and we can assist you.

The experiment runner script requires the following python libraries be installed:

- uhexp
- paramiko

1.2.3 Output Analysis (`nflogalyze.py`)

The analysis script requires the following python libraries to be installed:

- matplotlib

1.2.4 Documentation

This documentation can be rebuilt out of the primary repository.

Should you choose to do so you will need the following python libraries and tools:

- sphinx
- sphinx-rtd-theme

If you choose to produce PDF documentation you may also need `latex` packages for your system.

1.3 Experiment Quickstart

There is detailed documentation of all of the options available for each script available later this this documentation. This section is a quick example of how to run the basic experiment with minimal non-default parameters (and limited explanation).

1.3.1 Install Tools

You will need to install a number of python libraries - if you are not using a virtualenv you may have to use `sudo` with `pip`:

```
$ pip install --upgrade requests lxml cryptography ipaddress paramiko matplotlib
```

You will need a working *geni-lib* installation set up with your GENI credentials. You can find installation instructions in the online geni-lib documentation.

You will also need to clone and install a pair of libraries provided by the UH NetLab:

```
$ hg clone https://bitbucket.org/uh-netlab/uhgeni
$ hg clone https://bitbucket.org/uh-netlab/uhexp

$ cd uhgeni; pip install .; cd ..
$ cd uhexp; pip install .; cd ..
```

Finally, we will clone the repository containing the tools to build the experiments:

```
$ hg clone https://bitbucket.org/uh-netlab/doe-nf
$ cd doe-nf
```

1.3.2 Run Experiment

In order to run the experiment you need to choose a VTS site to use with `scripts/reserve.py` - we'll use the site located at GPO for this example, but you can choose any publicly-available VTS site. Once you have chosen a site you will need to link a Dropbox account with your user credential at that site, in order to transfer data for analysis out of your isolated topology, as VTS topologies do not have access to the internet. You can find instructions for this in the online GENI VTS documentation. The experiment will still function without this feature, but you will not be able to easily retrieve data for analysis.

When you are ready to execute the experiment, you will make a reservation and then use the reserved topology details to invoke the experiment orchestration script (replace `myslicename` with a slice you have created):

```
$ scripts/reserve.py --delete --slice myslicename --site vts-gpo --num-sites 2 --num-
↪sensors 2 --with-nf --mgmt-delay 1000 --mgmt-reorder 20
```

This will create a topology with two sensor sites that have two sensors each, 1000ms of delay (typical of geosynchronous satellite transport), 20% packet reordering, and employ the NF in the topology. It will also delete any previously existing sliver at this site if one exists. Once this returns it will have saved the manifest, request rspec, and *dot* file representing your reservation in the current directory. You can now run the actual experiment:

```
$ scripts/runexp.py --sites 2 --sensors 2 --sensor-rate 25 --run-time 30
```

You will be asked for the passphrase for any SSH private keys needed to execute the experiment. After setting up all the nodes in the topology this will run the sensors for 30 seconds and then shut everything down. You can view the current progress by tailing the live output log in another terminal:

```
$ tail -F doe-exp.log
```

1.3.3 Generate Charts

If you have enabled Dropbox syncing with the VTS site, your runtime data will appear in your dropbox in `Apps/vts-gpo/<sliver-uuid>` as the experiment run ends.

If your Dropbox folder is not accessible from the place where you installed the analysis tools, you will need to copy `nf2/nf2/_host_root/nf-eth1-eth2.log` to an accessible location. You can generate a standard set of charts using the following command:

```
$ scripts/nflogalyze.py --logs /path/to/nf-eth1-eth2.log
```

By default this will generate charts in PNG format in the current directory.

Note: Your system may need to have *tk* installed and the instructions vary per OS and distribution. On Ubuntu you can typically execute `sudo apt-get install python-tk` and it will install the system libraries you need.

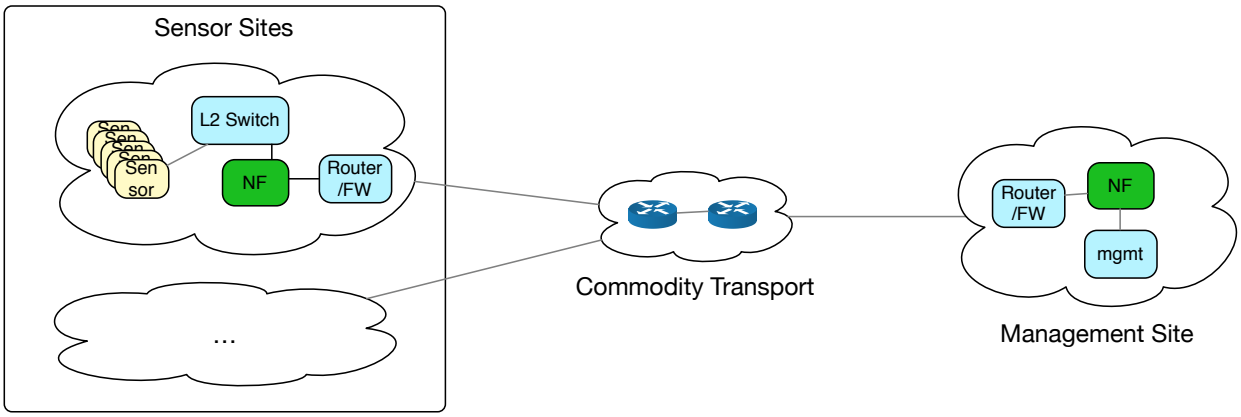
CHAPTER 2

Experiment Script Documentation

2.1 Experiment Overview

The topology creator (`reserve.py`) and experiment runner (`runexp.py`) in this repository create a topology and run a limited experiment on it. The root repositories for this project will contain more advanced experiments over time. This limited experiment evaluates the effect of packet reordering on packet delivery time, and also resource usage in the network function (queue depth, and thus memory usage).

The `reserve.py` script creates a single remote management endpoint and an arbitrary number of remote sites with a configured number of sensors for those sites. Packets between these sites traverse a commodity transport that is unreliable and that unreliability is configurable (latency and reordering). The basic topology configuration is as shown below:



You can configure the number of packets-per-second (pps) sent by each sensor, the number of sites and sensors, the latency in the network, and the reordering applied to those packets. The default GENI VTS reordering is applied using netem, which sends packets *before* their typical delay, so that delay is broken up across the network to make the mean delivery time still match your requested latency while offering reordering.

Note: The reordering applied delay is by default 10% of the overall requested delay, which means that the interarrival time from the sensors must be less than the delay in order to cause reodering in the same flow (e.g. if 10% of your delay is 100ms and you send packets twice a second, reordering within the same flow will not occur).

2.2 Topology Reservation

`reserve.py` takes a number of arguments, most of which are optional.

`--site`	Site to deploy topology, as a site name (vts-gpo, etc.)
`--slice`	Slice name
`--project`	Project name (typically the geni-lib default)
`--with-nf`	Build with network function – don't set to get baseline
`--context-path`	Path to context JSON, if not default location
`--delete`	Delete any pre-existing sliver with same slice name
`--num-sites`	Number of sites to deploy in topology
`--num-sensors`	Number of sensors to deploy per site
`--mgmt-loss`	Percentage of loss in network
`--mgmt-delay`	Delay in ms between every site and mgmt location
`--mgmt-reorder`	Percent of packets to reorder
`--config-path`	SSH Config file location for constructed topology
`--uhexp-url`	URL for `uhexp` repository
`--uhexp-branch`	Branch in supplied repository
`--nf-url`	URL for `doe-nf` repository
`--nf-branch`	Branch in supplied repository
`--nfbase-url`	URL for `pynf-base` repository
`--nfbase-branch`	Branch in supplied repository

Only `slice` is a required argument if you have a properly set up `geni-lib` context. You can use the URLs and branches to supply your own modification to the experiment. The default values of `num-sites` is 2 and `num-sensors` is 3, which you will need to know to provide input to `runexp.py`. You will have to set reorder and delay parameters in order to get useful results, but they are not required for basic packet delivery to function.

The common execution of `reserve.py` for an NF deployment will simply be:

```
$ reserve.py --with-nf --mgmt-delay 1000 --mgmt-reorder 20 --slice slicename
```

2.3 Experiment Execution

`runexp.py` takes a small number of arguments, with some required based on the values passed to `reserve.py`.

`--sites`	Number of sites in the topology
`--sensors`	Number of sensors per site
`--sensor-rate`	Number of packets per second sent by each sensor
`--debug`	Enable debug logging
`--setup-only`	Only setup experiment, without running
`--run-time`	Duration to run sensors and NF in experiment

`runexp.py` must be passed information about the topology, as it cannot determine the values without assistance. A typical run of the experiment for 30 seconds would be:

```
$ runexp.py --sites 2 --sensors 3 --run-time 30
```

2.4 Data Analysis and Chart Creation

The experiment orchestrator by default creates a robust amount of validation data (if you use `--debug` more data will be saved that are not typically useful unless you are experiencing runtime issues). *pcap* files are stored for each interface on each Network Function (NF), and the NF itself writes a log recording packet actions (stored in `_host_root/nf-eth[x]-eth[y].log` for each NF host). The NF log is used for primary evaluation and chart generation, while the *pcap* files are typically used for manual validation of unusual results.

`nflogalyze.py` parses the NF log and generates charts for either a single run, or panels for a set of runs. You can control the output with the following options:

`--logs`	List of logfiles for which to provide charts
`--panels`	Whether to generate panels for multiple log inputs
`--num-rows`	Number of rows to use for the panel charts
`--num-cols`	Number of columns to use for the panel charts
`--type`	Image file type

Typical operation for generating charts for one experiment run would be:

```
$ nflogalyze.py --logs nf-eth1-eth2.log
```

For multiple runs the number of rows and columns need to multiply to the total number of logs provided for the constituent runs. For 10 runs you could use the following command:

```
$ nflogalyze.py --logs *.log --panels --num-rows 2 --num-cols 5
```

The logs must be named in the form `[*]XX.log` where `XX` is the reorder percentage in order to get sensible chart titles.