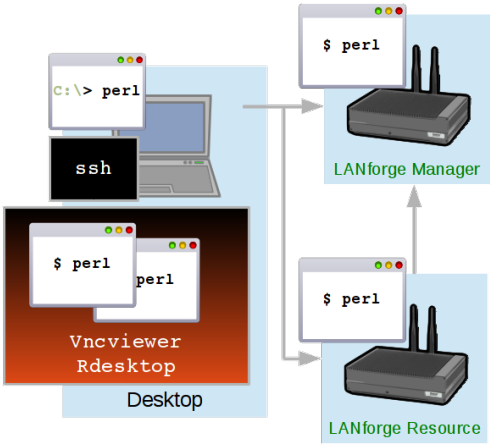
sales@candelatech.com
support@candelatech.com
+1 (360) 380-1618 [PST, GMT -8]

LANforge Scripting Cookbook

The LANforge Scripting Cookbook provides a series of detailed examples of how to craft testing scripts for unattended/automated operation. Each example intends to give the reader a runnable test script and a better understanding of how to use the LANforge Scripting.

Places to Run CLI Commands

You do not need to operate scripts directly from the LANforge server, and this allows you to code scripts in your preferred text editing environment. Likewise, you do not need to run a copy of LANforge Server on your desktop. Scripts will create a plain-text connection to the LANforger server you specify.



Windows Desktop

You can install a copy of the LANforge Server on your windows desktop (without a license) so that you have access to the Perl scripting libraries. Edit scripts and run them from your `C:\Program Files\LANforge-Server\scripts` directory.

Linux Desktop

You can copy the LANforge scripts folder directly from your LANforge server to your Documents directory with scp.

SSH or VNC connection to LANforge Server

Using *vncviewer*, *rdesktop* or *ssh* are all fine options to connect to the LANforge server to write and operate scripts. The LANforge server comes with a basic Linux desktop and you can use emacs, vim, pluma, or gedit text editors installed by default. When editing scripts on the LANforge server itself, be careful to back up your work before you upgrade LANforge. The LANforge install process will over-write scripts of the same name in the scripts directory.

Requirements for Scripts

Your desktop (or other computer) running CLI scripts needs to have a reliable (wired) connection to the management port of your LANforge server. If you are engaging in long running tests, you might consider running the scripts from the LANforge manager itself if your desktop machine needs to be powered off.

Script Libraries

CLI scripts are written using Perl. They require the libraries in `/home/lanforge/scripts/LANforge` Users may write scripts in other programming languages, such as python, but in that case, they will not be able to take direct avantage of the Perl scripts included in LANforge.

On Windows

LANforge is more fully featured on Linux, but basic support exists on Windows as well.

You can run CLI scripts from any Windows desktop as long as you have Perl installed. You can use ActiveState Perl or Perl from the Cygwin project. We also highly suggest installing PuTTY ssh client to access your LANforge server.

On Linux/OS X

Most Linux distributions come with and ssh client and Perl already installed.

LANforge Server Requirements

The following examples will create test scenarios that work on LANforge Linux systems running the LANforge software with the LANforge kernel and a sufficient license. If you are running LANforge server using another Linux kernel, you may not be able to operate some of the examples. (Features like Armageddon, operation of WiFi-AC radios, and WanLinks all require drivers included only in Candela provided kernels.)

Please contact us at support@candelatech.com if you have any questions.

Before Starting LANforge-CLI Traffic Generation

Before attempting the examples below, ensure that you have successfully followed these software installation guides:

- LANforge-GUI Installation
- LANforge Server Installation

It is also recommended that you back up your current running LANforge Server database so that you may safely return to your current operating state.
For instance:

```
su - root
```

```
cd /home/lanforge
tar -cvzf my_db_backup.tar.gz DB
```

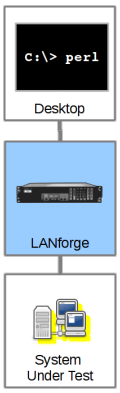
LANforge-CLI Detailed Cookbook Examples

1. Perl: Perl CLI Scripts Introduction
2. Perl: Operating LANforge scripts from Windows
3. Perl: Monitor and Reset Ports with the portmod Script
4. Perl: Cross Connects and Endpoints Tutorial
5. Perl: Creating Connections with the FIREmod Script
6. Perl: Operating Endpoint Hunt Scripts with the CLI
7. Perl: Generating WiFi Traffic with the Associate Script
8. Perl: Changing Station WiFi SSID with the CLI API
9. CLI: Changing Station POST_IFUP Script with the CLI API
10. CLI: Scripting Attenuation with CSV data
11. Basics: LANforge Entity IDs
12. Basics: LANforge GUI Introduction
13. JSON: Querying the LANforge Client for JSON Data
14. CLI: Station CLI Operations
15. Perl: Generic Endpoint Scripts
16. Perl: Chamber View: Automated tests with script
17. Basics: LANforge Scripting Introduction
18. Python: Querying the LANforge JSON API using Python
19. Python: Managing WANlinks using JSON and Python
20. Perl: Control a chamber with the lf_chamber.pl Script
21. Perl: Emulating Video Transmission with Layer 3 connections
22. Python: Create Test Scripts With the Realm Class
23. Python: Create Layer 4 Test Scripts With Python
24. Python: Create Generic Test Scripts With Python
25. Python: Create VAP Test Scripts With Python
26. Python: Load Scenarios And Control Test Groups With Python
27. Python: Record the results of a test as CSV from the REALM monitor script
28. Python: Record the results of a test as an Excel file from the REALM monitor script
29. Python: Define and Demonstrate Docstring Usage in Candelatech Python Scripts
30. Python: Scan for SSIDs, BSSIDs, and Signals of wireless APs
31. Python: Probe Ports for Information
32. Suite: Start Here: Initial Setup to Run Scripts Test Suite for AP Testing
33. Getting Started: A Simple Script: sta_connect2.py
34. Getting Started: Basic: Layer 3 Traffic Generation: test_l3.py
35. Getting Started: Basic: Layer 4-7 HTTP Traffic Generation: test_l4.py
36. Getting Started: Basic: Layer 4-7 FTP Traffic Generation: test_l4.py
37. Getting Started: Configure and Run Dataplane Test
38. Multiplexed REST Access via Nginx Proxy

Introduction to CLI Scripts

Goal: You will be able to execute LANforge testing scripts from the command line from Windows, Linux, remote desktop or ssh connection.

Traffic emulation may be run unattended and automated using Perl scripts provided with the LANforge Server. These scripts can be run from within the LANforge server or outside the LANforge Server (on a Windows or other desktop). The output of the scripts should be redirected into a text file for you to process the results.



Where Do I Find Scripts?

On Windows

On most versions of windows, the LANforge Server installs scripts in

```
C:\Program Files (x86)\LANforge-Server\scripts
```

On Linux

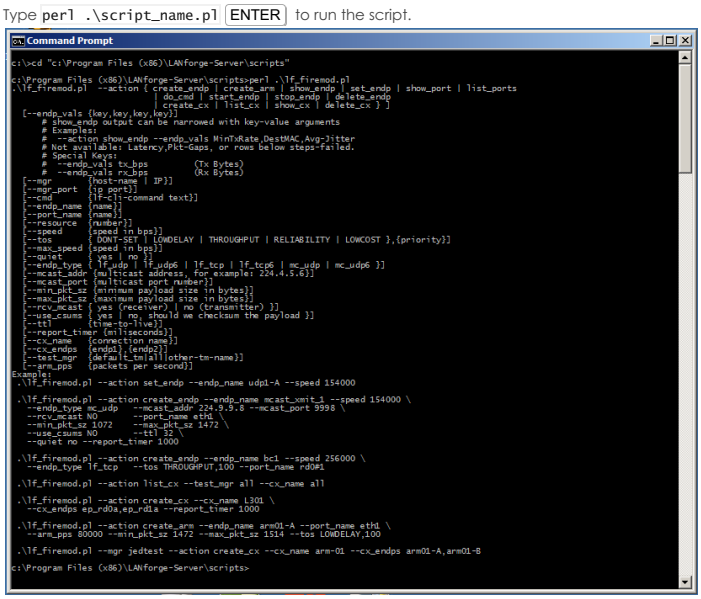
In the home directory for user LANforge:

```
/home/lanforge/scripts
```

How to Run Scripts

Starting a script on Windows:

1. Make sure that perl is in your `PATH`. (See Inspecting DOS PATH)
2. Open a CMD window (or a PowerShell window)
3. and change directory to `C:\Program Files (x86)\LANforge-Server\scripts`
4. Type `perl .\script_name.pl` ENTER to run the script.



Generally, the script will tell you that it needs more switches.

Finding Help

Most scripts have a `-h` or `--help` switch that explain what switches they expect.

Script Conventions

In general, scripts will expect you to tell them a few things, regardless of the script:

- The manager IP address to connect to:`--mgr 127.0.0.1` or `--manager 192.168.100.1`. This often defaults to 127.0.0.1, but when connecting from outside the machine, please use the IP of the LANforge management port (often `eth0`).
- The manager port to connect to:`--port 4001` or `--mgr_port 4001`. This often defaults to 4001.
- Which resource to direct the command to. The manager is always resource 1. Resource 2 would be your second LANforge server. `--resource 2`. Some scripts use the older term **card**: `--card`
- If you need debugging output, turn off *quiet* mode: `-q no` or `--quiet no`. Some older scripts want you to turn quiet on explicitly: `--quiet 1`
- To capture the output, use the `>` operator to redirect the text output into a text file.
- Scripts are often executed from within a shell script (or batch file). Often the formatting of the commands includes '\' characters which indicate 'continue this command on the next line of input.' Here is an example of formatting a single script command on multipl lines:

```
$ ./lf_portmod.pl \
  --manager 192.168.100.1 \
  --card 3 \
  --show_port
```

- Comments begin with '#'. They are lines ignored by the shell, and they are also comments in perl.

Running on local LANforge manager

You can use ssh, VNC or Rdesktop to connect from your desktop to your LANforge manager server. (When using VNC, assume display `:1`). From there, in a terminal, you will execute your script from the `/home/lanforge/scripts` directory as shown in the example below:

```
 $ cd /home/lanforge/scripts
 $  ./lf_portmod.pl --help
#...help appears...
 $ ./lf_portmod.pl --manager 192.168.100.1 --card 1 --show_port
# ... displays port info
```

Running on local LANforge resource

If you connect to a LANforge resource and want to run a script, you must direct the script at the LANforge manager server and specify the resource you are interested in. For example, you might be on resource 2 (192.168.100.2) and desire to run tests on resource 3 (192.168.100.3):

```
$ cd /home/lanforge/scripts
$  ./lf_portmod.pl --manager 192.168.100.1 --card 3 --show_port
# ... displays port info
```

Running on a Linux Desktop to a Remote LANforge

A more detailed set of steps follows. When running LANforge CLI scripts on a Linux desktop, you normally want to download and un-zip a copy of the LANforge-Server install file found on the Candela Technologies downloads page. Use a link similar to: `http://www.candelatech.com/private/downloads/r5.3.2/LANforgeServer-5.3.2_Linux-F21-x64.tar.gz`. For best results, use the scripts packaged with the version of LANforge to which your scripts will be connecting.

1. Open a terminal on your desktop, cd to your Downloads folder
2. use `wget` or `curl` to download the tar file:

   ```
   wget "http://guest:guest@www.candelatech.com/private/downloads/r5.3.2/LANforgeServer-5.3.2_Linux-F21-x64.tar.gz"
   ```

3. Create a scripts directory in your Documents folder:

   ```
   $ cd ~/Documents/scripts
   ```

4. Expand the tar file in your Downloads directory:

   ```
   $ tar xf LANforgeServer*tar.gz
   ```

5. Copy the scripts file into your Documents folder:

   ```
   $ cp -r LANforgeServer-5.3.2/scripts/. ~/Documents/scripts/
   ```

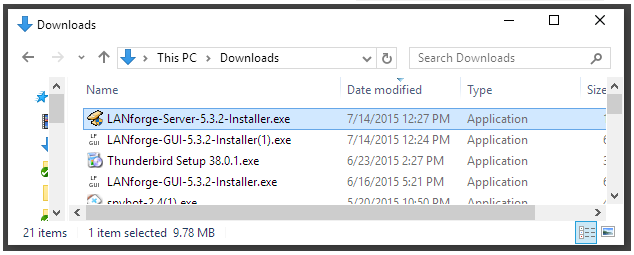
To use your scripts, in your terminal, change directories to `~/Documents/scripts` and they will operate similar to the above examples.

```
$ cd ~/Documents/scripts
$ ./lf_portmod.pl --manager 192.168.100.1 --card 3 --show_port
```

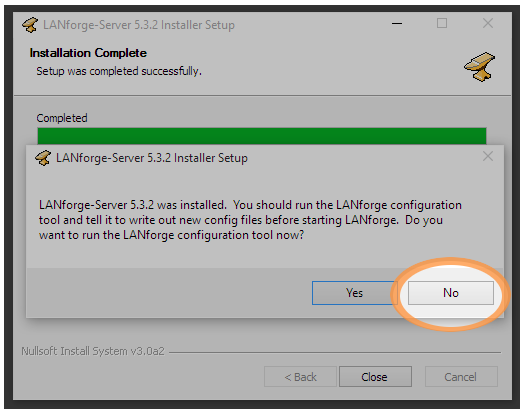
Running on a Windows Desktop to a Remote LANforge

The process for running CLI scripts on a Windows desktop is roughly similar, but involves running the Windows LANforge Server installer. This process does not require a Windows license as we will not be running the windows LANforge server. Perl is required to run Windows scripts. Start by installing that. You can use the perl that comes with the Cygwin project or if you just want perl, install the ActiveState ActivePerl package. ActivePerl should install update your environment `%PATH%` variable. If it does not immediately, you might need to log out and log back in.

1. Download the Windows version of the LANforge Server installer using your browser: `http://www.candelatech.com/private/downloads/r5.3.2/LANforge-Server-5.3.2-Installer.exe`. Use username *guest*, password *guest*.
2. In your Downloads folder, double click and install the `LANforge-Server-Installer.exe`.

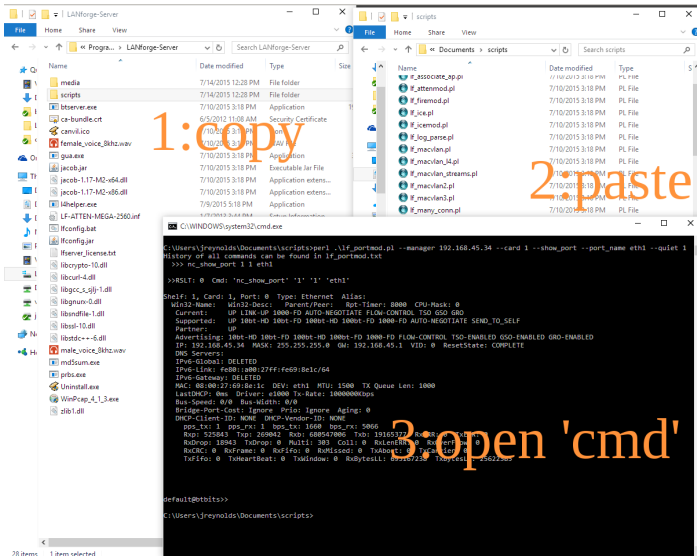
   

   Do not configure it, do not run LANforgeServer.



You will not need to be running the LANforge GUI to do this install.

3. The installation *scripts* folder will be system-protected, so you want to copy the folder over to your desktop *Documents* directory.

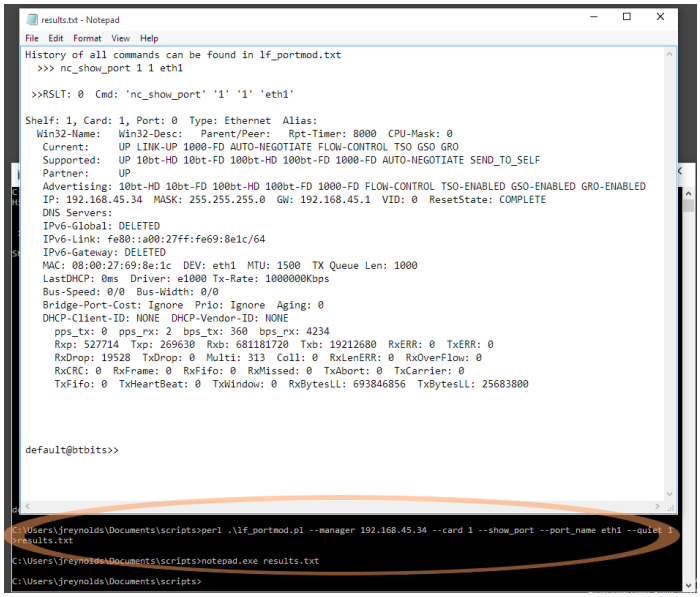


4. Open a DOS terminal, either using *Run→cmd* Enter or *Run→powershell* Enter
5. Change to the new copy of the scripts directory, and then you can run scripts by giving them to perl:

```
C:> cd C:\Users\bob\Documents\scripts
C:> perl .\lf_portmod.pl --manager 192.168.100.1 --card 3 --show_port --port_name eth1 --quiet 1
```

6. To capture output from a script, use the shell redirect operator: >. This example shows redirecting and browsing the results with *Notepad*:

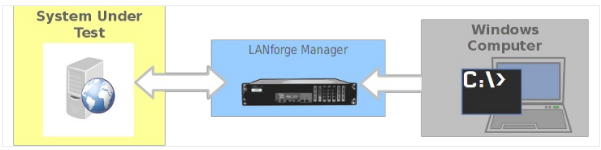
```
C:> perl .\lf_portmod.pl --manager 192.168.100.1 --card 3 --show_port --port_name eth1 --quiet 1 > results.txt
C:> notepad.exe results.txt
```



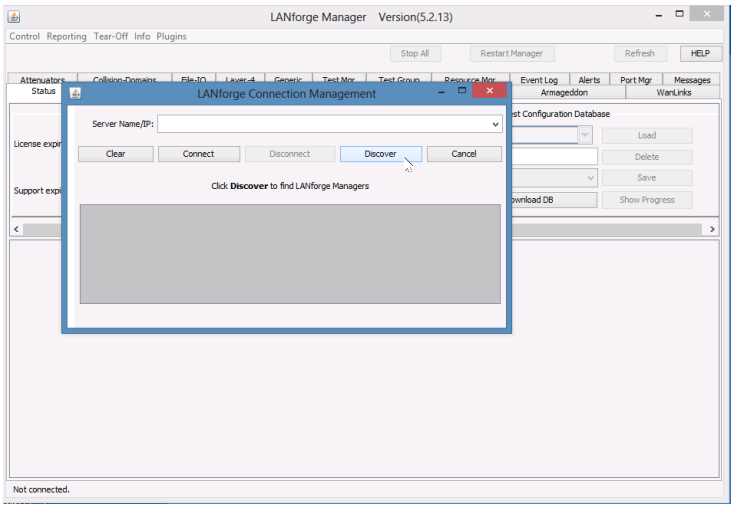
Operate LANforge Scripts from Windows.

Goal: Use an installation of LANforge and Perl on a Windows computer to operate tests and manage connections on remote LANforge computers.

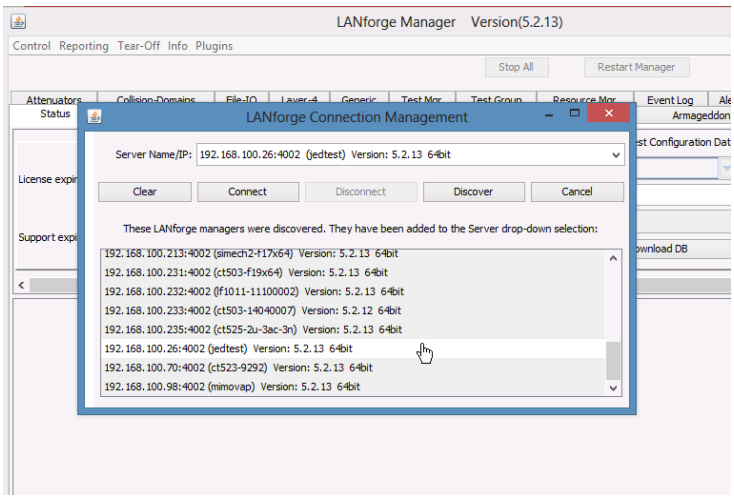
You do not need to connect to your LANforge manager using ssh or VNC to manage connections or operate tests. In this cookbook, you will see an example of using the `lf_firemod.pl` script which can give you port information. This example will require a Windows desktop with Java, ActiveState Perl, the LANforge Server and the LANforge GUI installed. You will not need to start the LANforge server on your Windows computer, so no licenses will be required for operating on the Windows computer. This cookbook assumes connectivity between the Windows computer and a running LANforge manager computer.



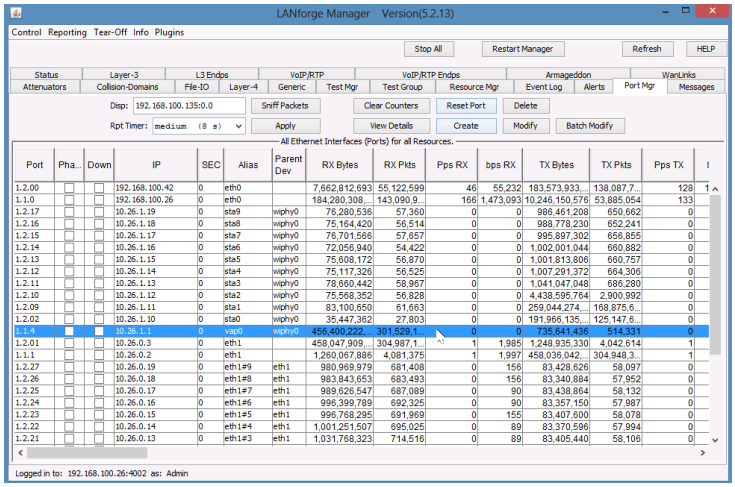
1. Prepare your Windows computer:
2. Install LANforge Server
   A. You do not need to configure or start this server. Only the perl scripts directory of this installation will be used.
   B. You can download it from our current releases page. For more information see LANforge Server Installation
3. Install LANforge GUI
   A. You can download it from the same location.
   B. Make sure you can connect to your LANforge manager. In this example, the LANforge manager will be at `192.168.100.26`
   C. Open the LANforge GUI



   D. ... and click **Discover**



E. If you are able to connect you should be able to browse your ports and connections.

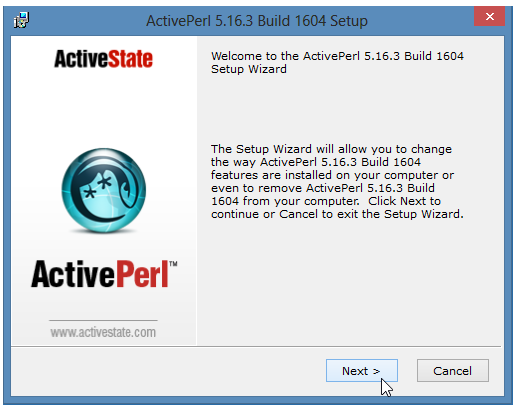


For more information see LANforge GUI Installation

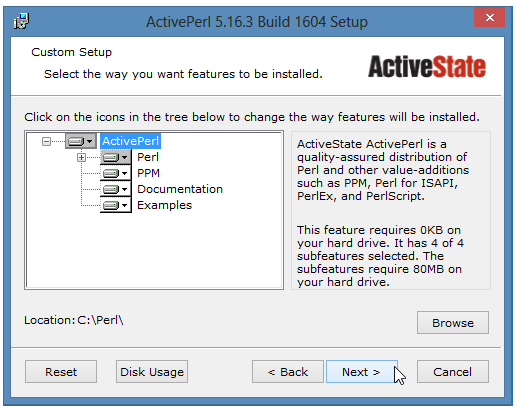
4. Install ActiveState Perl

   A. Please download it from the ActiveState downloads page.

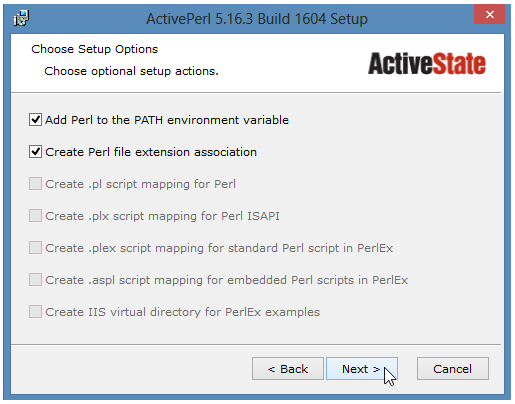
   B. Begin and press next...



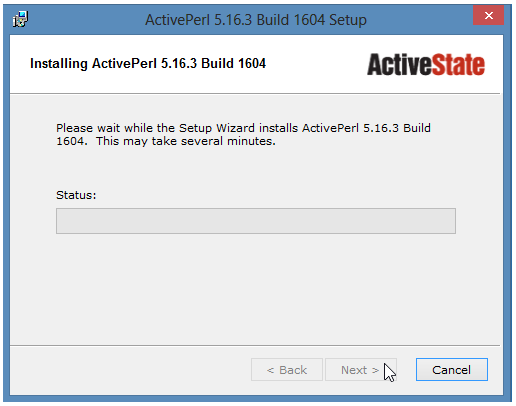
   C. ...press next...



   D. ...press next...



E. ...and then wait a few minutes...



F. Now, you are done, press **Finish**.

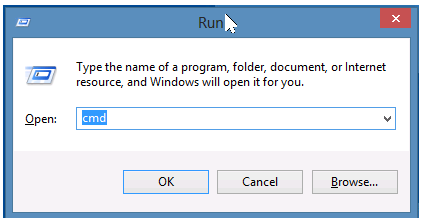
G. For advanced GUI scripting in Windows, you may also wish to view the Win32::GuiTest perl module page.

5. Using scripts from Windows

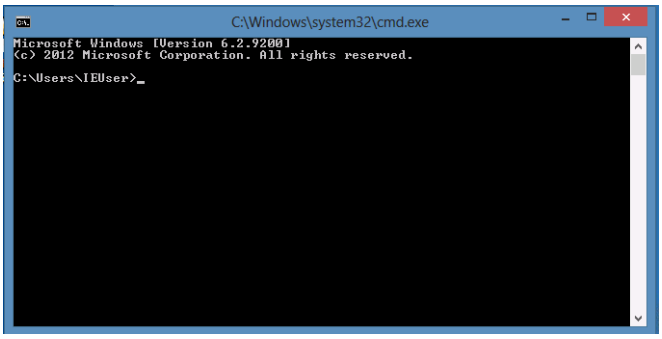
A. The scripts installed on your Windows computer will communicate with the LANforge manager over the management port (TCP 4001).

B. Open a **cmd** window. Click **Start->Run**, type `cmd` and press **Enter**

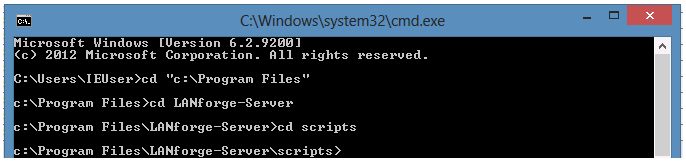
A.



B.



C. Change directory to `C:\Program Files\LANforge-Server\scripts`



D. List the script files: `dir *.pl`

```
c:\Program Files\LANforge-Server\scripts>dir *pl
 Volume in drive C has no label.
 Volume Serial Number is B6DE-1E48

 Directory of c:\Program Files\LANforge-Server\scripts

08/07/2014  04:32 PM             3,205 ftp-upload.pl
08/07/2014  04:32 PM            32,537 lf_associate_ap.pl
08/07/2014  04:32 PM             3,163 lf_attenmod.pl
08/07/2014  04:32 PM             7,370 lf_firemod.pl
08/07/2014  04:32 PM            10,162 lf_ice.pl
08/07/2014  04:32 PM             4,322 lf_icemod.pl
08/07/2014  04:32 PM               419 lf_log_parse.pl
08/07/2014  04:32 PM            44,633 lf_macvlan.pl
08/07/2014  04:32 PM            15,644 lf_macvlan2.pl
08/07/2014  04:32 PM            17,240 lf_macvlan3.pl
08/07/2014  04:32 PM            19,122 lf_macvlan_14.pl
08/07/2014  04:32 PM            17,419 lf_macvlan_streams.pl
08/07/2014  04:32 PM            13,830 lf_many_conn.pl
08/07/2014  04:32 PM             8,557 lf_monitor.pl
08/07/2014  04:32 PM            18,031 lf_netoptics.pl
08/07/2014  04:32 PM            32,819 lf_nfs_io.pl
08/07/2014  04:32 PM            11,610 lf_portmod.pl
08/07/2014  04:32 PM             8,191 lf_port_walk.pl
08/07/2014  04:32 PM             6,751 lf_stress1.pl
08/07/2014  04:32 PM             6,038 lf_stress2.pl
08/07/2014  04:32 PM             9,939 lf_stress3.pl
08/07/2014  04:32 PM             6,145 lf_stress4.pl
08/07/2014  04:32 PM            24,688 lf_verify.pl
08/07/2014  04:32 PM            22,242 lf_voip.pl
               24 File(s)        344,077 bytes
                0 Dir(s)  117,004,115,968 bytes free

c:\Program Files\LANforge-Server\scripts>_
```

E. Your installation of Perl should have put it into your path variable (%PATH%). Please verify that it did with this command: `perl -v`

```
C:\Windows\system32\cmd.exe

c:\Program Files\LANforge-Server\scripts>perl -v

This is perl 5, version 16, subversion 3 (v5.16.3) built for MSWin32-x86-multi-t
hread
(with 1 registered patch, see perl -V for more detail)

Copyright 1987-2012, Larry Wall

Binary build 1604 [298023] provided by ActiveState http://www.ActiveState.com
Built Apr 14 2014 14:32:20

Perl may be copied only under the terms of either the Artistic License or the
GNU General Public License, which may be found in the Perl 5 source kit.

Complete documentation for Perl, including FAQ lists, should be found on
this system using "man perl" or "perldoc perl".  If you have access to the
Internet, point your browser at http://www.perl.org/, the Perl Home Page.


c:\Program Files\LANforge-Server\scripts>_
```

F. If perl is not found (`Command not found`) then you might need to close your DOS window and open a new one, or your Windows computer might need a reboot for the `PATH` variable to take effect.

G. Start the `lf_firemod.pl` script with the **--help** switch to see the options.

```
C:\Windows\system32\cmd.exe

c:\Program Files\LANforge-Server\scripts>perl lf_firemod.pl --help
Unknown option: help
lf_firemod.pl  --action { show_endp | set_endp | show_port | list_ports | do_cmd
 } ]
                [--endp_vals {key,key,key,key}]
                    # show_endp output can be narrowed with key-value arguments

                    # Examples:
                    # --action show_endp --endp_vals MinTxRate,DestMAC,Avg-Jitt
er
                    # Not available: Latency,Pkt-Gaps, or rows below steps-fail
ed.
                    # Special Keys:
                    #  --endp_vals tx_bps          (Tx Bytes)
                    #  --endp_vals rx_bps          (Rx Bytes)
                [--mgr {host-name | IP}]
                [--mgr_port {ip port}]
                [--cmd {lf-cli-command text}]
                [--endp_name {name}]
                [--port_name {name}]
                [--resource {number}]
                [--speed {speed in bps}]
                [--quiet { yes | no }]

Example:
 lf_firemod.pl --action set_endp --endp_name udp1-A --speed 154000

c:\Program Files\LANforge-Server\scripts>_
```

A. `perl lf_firemod.pl --help`

H. Open a second `cmd` window so that you can see the help text in the first window. Change directory to `C:\Program Files\LANforge-Server\Scripts`

I. Use this command to list ports available on `192.168.100.26`:

A. `perl lf_firemod.pl --mgr 192.168.100.26 --resource 1 --action list_ports`

B.

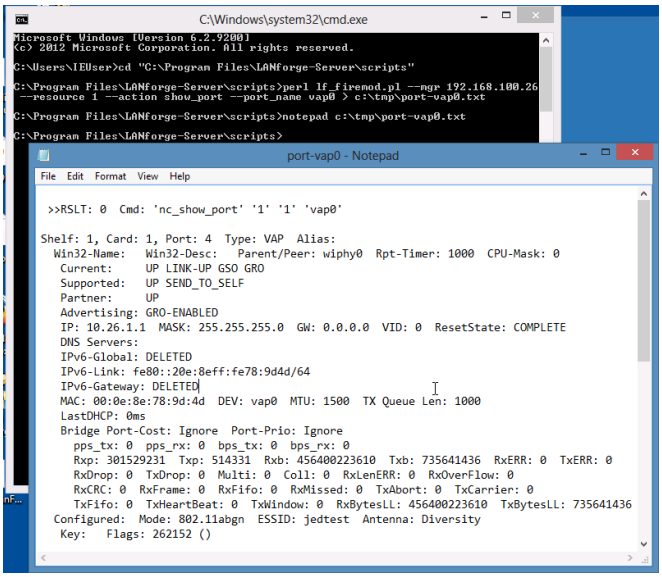
```
C:\Windows\system32\cmd.exe

C:\Program Files\LANforge-Server\scripts>perl lf_firemod.pl --mgr 192.168.100.26
 --resource 1 --action list_ports
eth1 link=UP speed=1G
wiphy0 link=DOWN speed=UNKNOWN
eth0 link=UP speed=1G
wiphy1 link=DOWN speed=UNKNOWN
vap0 link=UP speed=UNKNOWN

C:\Program Files\LANforge-Server\scripts>_
```

J. Most command output shows considerably more text than the output of the previous command. You may want to pipe it to a file. In this example, the output is redirected to `C:\tmp\port-vap0.txt` and shown with Notepad.

A. Query the port stats using: `perl lf_firemod.pl --mgr 192.168.100.26 --resource 1 --action show_port --port vap0 > c:\tmp\port-vap0.txt`

```
C:\Windows\system32\cmd.exe

Microsoft Windows [Version 6.2.9200]
(c) 2012 Microsoft Corporation. All rights reserved.

C:\Users\IEUser>cd "C:\Program Files\LANforge-Server\scripts"

C:\Program Files\LANforge-Server\scripts>perl lf_firemod.pl --mgr 192.168.100.26
 --resource 1 --action show_port --port_name vap0 > c:\tmp\port-vap0.txt

C:\Program Files\LANforge-Server\scripts>
```

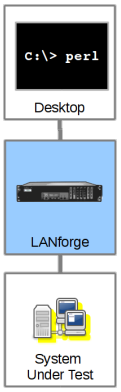
B. Show the output with: `notepad c:\tmp\port-vap0.txt`



Inspecting Ports (Network Interfaces) using lf_portmod

Goal: You will be able to report and reset ports on your LANforge server.

Port statistics can be programatically monitored using the script `lf_portmod.pl`. This script can also reset ports, alter WiFi station settings, and pass arbitrary LANforge CLI commands directly to the LANforge manager.



Ports of all kinds can be viewed with the **lf_portmod.pl** perl script. You can also do some limited manipulation of ports as well.

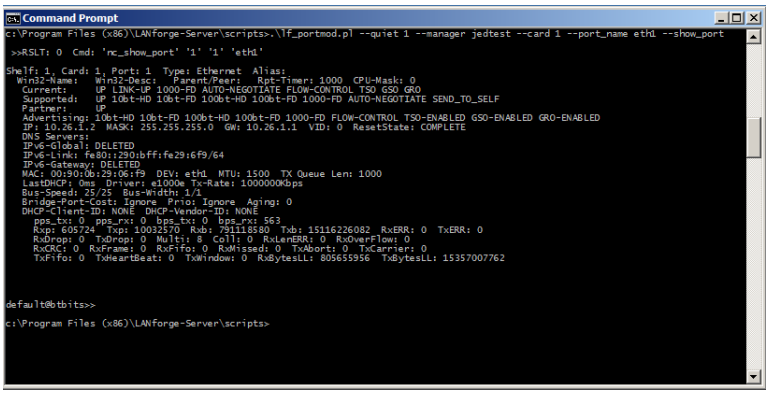
Listing Ports

You can show statistic on a port with the `--show_port` argument:

```
C:\> perl .\lf_portmod.pl --quiet 1 --manager jedtest --card 1 --port_name eth1 --show_port
```

i You can right-click to paste these commands into your DOS window

Produces:

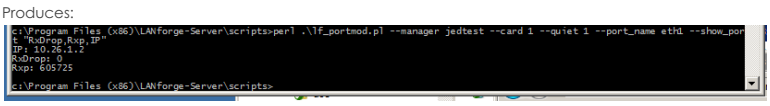


Listing Port Attributes

Individual port attributes can also be shown, which often makes automating reporting easier.

```
perl .\lf_portmod.pl --manager jedtest --card 1 --quiet 1 --port_name eth1 --show_port "RxDrop,Rxp,IP"
```

Produces:

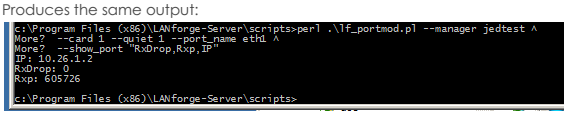


Consider that is a lot of text to type. If we want, we can reformat that command.

> i Long DOS commands and be continued on the next line with the ^ character.

```
perl .\lf_portmod.pl --manager jedtest ^
 --card 1 --quiet 1 --port_name eth1 ^
 --show_port "RxDrop,Rxp,IP"
```

Produces the same output:

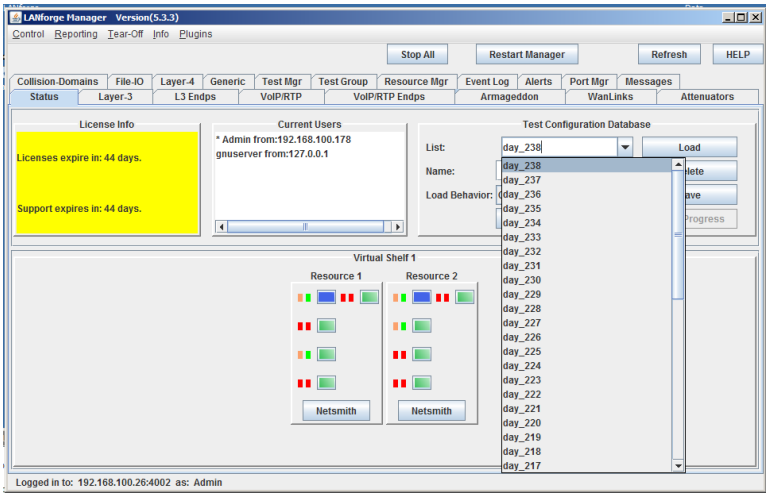


Loading a test scenario

Saved test scenarios are often referred to as 'databases'

```
lf_portmod.pl --load day_238
```

This matches the same database name seen in the *Status* tab database dropdown.



Admin-down a port

```
lf_portmod.pl --manager 192.168.1.101 --card 1 --port_name eth2 --set_ifstate down
```

Resetting a Port

Resetting a port forces a port to unload and reload its configuration.

```
lf_portmod.pl --manager 192.168.1.101 --card 1 --port_name eth2 --cmd reset
```

Sending a specific CLI command to the LANforge manager:

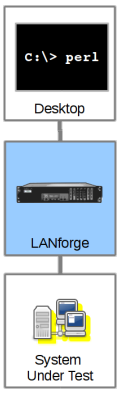
It is possible to directly pass a command to the LANforge manager:

```
lf_portmod.pl --manager 192.168.1.101 --cli_cmd "scan 1 1 sta0"
```

Cross Connects and Endpoints Tutorial

Goal: Gain a better understanding on how you will use Cross connects, Connections and Endpoints to use the LANforge CLI scripts knowlegably.

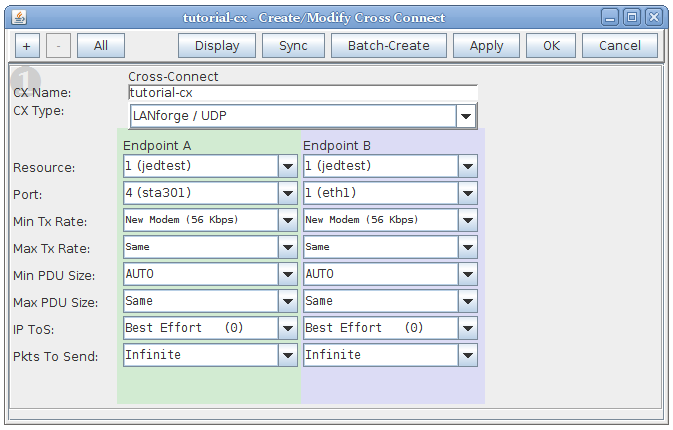
Creating connections in the LANforgeGUI implies creating endpoints. These endpoint entities are created with predictable names and are usually created in pairs. Understanding these naming conventions and how they are created is fundamental to your proficiency with creating connections with LANforge CLI scripts.



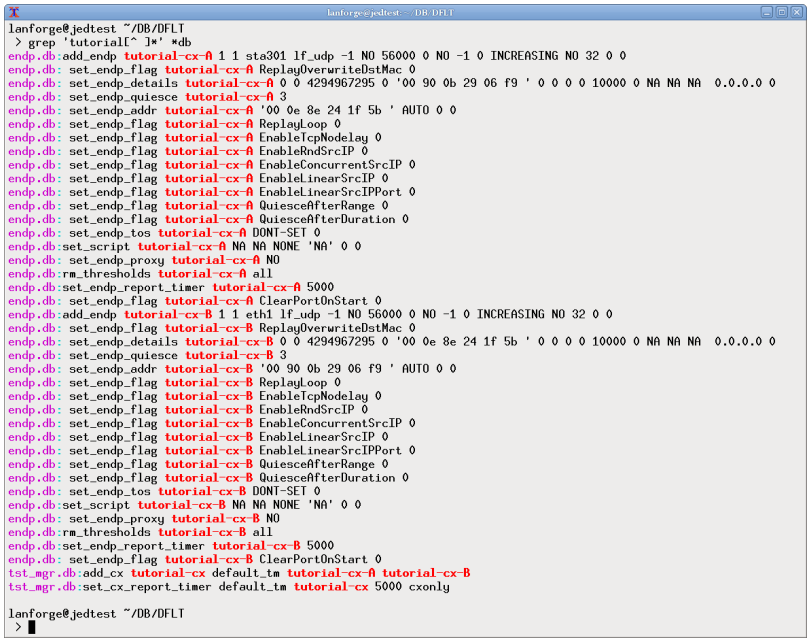
Most examples in our cookbooks assume a dual-ended connection, also known as a *cross-connect* or abbreviated as *CX*.

Building Endpoints and Connections

Let's follow the creation of a Connection:



Using a terminal on the LANforge machine, we look at the `/home/lanforge/DB/DFLT/endps.db` file and inspect the commands issued that create that connection:



```
lanforge@jedtest ~/DB/DFLT
 > grep 'tutorial[^ ]*' *db
endp.db:add_endp tutorial-cx-A 1 1 sta301 lf_udp -1 NO 56000 0 NO -1 0 INCREASING NO 32 0 0
endp.db: set_endp_flag tutorial-cx-A ReplayOverwriteDstMac 0
endp.db: set_endp_details tutorial-cx-A 0 0 4294967295 0 '00 90 0b 29 06 f9 ' 0 0 0 0 10000 0 NA NA NA  0.0.0.0 0
endp.db: set_endp_quiesce tutorial-cx-A 3
endp.db: set_endp_addr tutorial-cx-A '00 0e 8e 24 1f 5b ' AUTO 0 0
endp.db: set_endp_flag tutorial-cx-A ReplayLoop 0
endp.db: set_endp_flag tutorial-cx-A EnableTcpNodelay 0
endp.db: set_endp_flag tutorial-cx-A EnableRndSrcIP 0
endp.db: set_endp_flag tutorial-cx-A EnableConcurrentSrcIP 0
endp.db: set_endp_flag tutorial-cx-A EnableLinearSrcIP 0
endp.db: set_endp_flag tutorial-cx-A EnableLinearSrcIPPort 0
endp.db: set_endp_flag tutorial-cx-A QuiesceAfterRange 0
endp.db: set_endp_flag tutorial-cx-A QuiesceAfterDuration 0
endp.db: set_endp_tos tutorial-cx-A DONT-SET 0
endp.db:set_script tutorial-cx-A NA NA NONE 'NA' 0 0
endp.db: set_endp_proxy tutorial-cx-A NO
endp.db:rm_thresholds tutorial-cx-A all
endp.db:set_endp_report_timer tutorial-cx-A 5000
endp.db: set_endp_flag tutorial-cx-A ClearPortOnStart 0
endp.db:add_endp tutorial-cx-B 1 1 eth1 lf_udp -1 NO 56000 0 NO -1 0 INCREASING NO 32 0 0
endp.db: set_endp_flag tutorial-cx-B ReplayOverwriteDstMac 0
endp.db: set_endp_details tutorial-cx-B 0 0 4294967295 0 '00 0e 8e 24 1f 5b ' 0 0 0 0 10000 0 NA NA NA  0.0.0.0 0
endp.db: set_endp_quiesce tutorial-cx-B 3
endp.db: set_endp_addr tutorial-cx-B '00 90 0b 29 06 f9 ' AUTO 0 0
endp.db: set_endp_flag tutorial-cx-B ReplayLoop 0
endp.db: set_endp_flag tutorial-cx-B EnableTcpNodelay 0
endp.db: set_endp_flag tutorial-cx-B EnableRndSrcIP 0
endp.db: set_endp_flag tutorial-cx-B EnableConcurrentSrcIP 0
endp.db: set_endp_flag tutorial-cx-B EnableLinearSrcIP 0
endp.db: set_endp_flag tutorial-cx-B EnableLinearSrcIPPort 0
endp.db: set_endp_flag tutorial-cx-B QuiesceAfterRange 0
endp.db: set_endp_flag tutorial-cx-B QuiesceAfterDuration 0
endp.db: set_endp_tos tutorial-cx-B DONT-SET 0
endp.db:set_script tutorial-cx-B NA NA NONE 'NA' 0 0
endp.db: set_endp_proxy tutorial-cx-B NO
endp.db:rm_thresholds tutorial-cx-B all
endp.db:set_endp_report_timer tutorial-cx-B 5000
endp.db: set_endp_flag tutorial-cx-B ClearPortOnStart 0
tst_mgr.db:add_cx tutorial-cx default_tm tutorial-cx-A tutorial-cx-B
tst_mgr.db:set_cx_report_timer default_tm tutorial-cx 5000 cxonly

lanforge@jedtest ~/DB/DFLT
 >
```

That's a lot of commands. We will point out what is particularly necessary when using our Perl scripts.

Endpoints and Connections Naming Convention

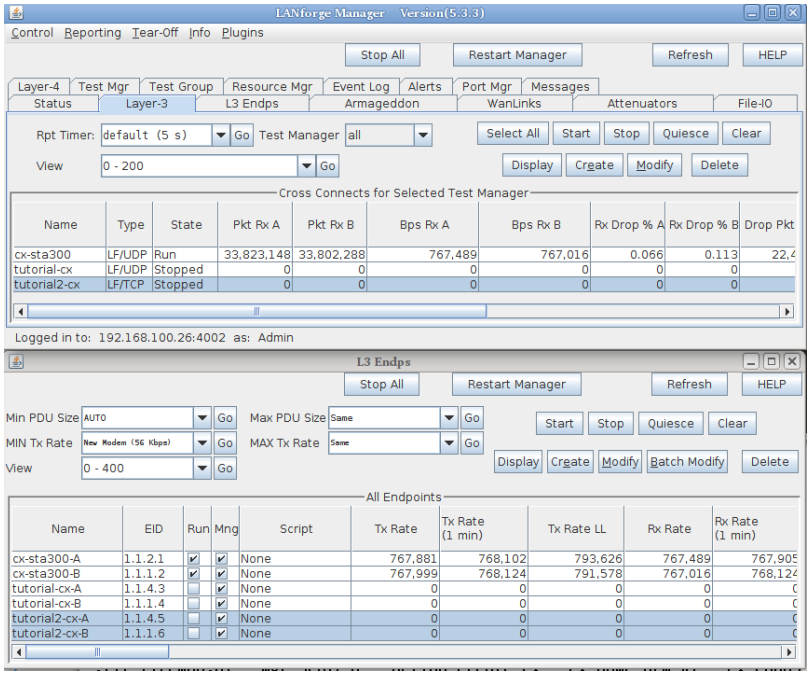
The connection we created above is named **tutorial-cx**. Two endpoints also have names, tutorial-cx-**A** and tutorial-cx-**B**. The A-side of a connection is always managed. A B-side endpoint may be unmanaged. When you write CLI scripts that create connections, name your endpoints using a similar convention.

Endpoints are Created First

We can use the lf_firemod.pl script to create endpoints and a cross connect in this order:

```
$ ./lf_firemod.pl --action create_endp --endp_name tutorial2-cx-A \
      --speed 256000 --endp_type lf_tcp --port_name sta301

$ ./lf_firemod.pl --action create_endp --endp_name tutorial2-cx-B \
      --speed 256000 --endp_type lf_tcp --port_name eth1

$ ./lf_firemod.pl --action create_cx --cx_name tutorial2-cx \
      --cx_endps tutorial2-cx-A,tutorial2-cx-B
```

We can see the results of those script commands in our **Layer-3** and **L3 Endps** tabs:



Multiple Windows can be displayed using the *Tear Offs* menu.

Clicking on the Layer-3 connection automatically highlights the two endpoints.

Starting and Stopping: Connections have State

When a connection is first created, it is STOPPED. When you start it, it becomes RUNNING. When you set a connection to STOPPED, both endpoints immediately stop sending and recieving. That can have a consequence of leaving unacknowledged packets in flight. The safest way is to QUIECE the connection, which first stops the endpoints transmitting, waits a short time, and then stops the endpoints from recieving.

When there is just one Endpoint

Normally, if you see one endpoint, it should only be a multicast endpoint. A single endpoint can be seen in these situations:

- You have paused between creating the first and second endpoint for a connection. Continue working.
- Created by a script mistakenly, through a typo or other misconfiguration
- Left over from an interrupted script that deleted the cross-connect and one of two endpoints

A single endpoint is not an illegal entity, but lonely endpoints can add confusion. If you find endpoints that do not match any existing connections, we suggest deleting them.

A Cross-Connect can be *one-sided*, that is, have one *unmanaged* endpoint. The A side endpoint is a LANforge managed port transmitting to another device that's not a LANforge machine. Some connection types create this style of endpoint pairs, like **File-endpoints** and **Layer 4-7 connections**.

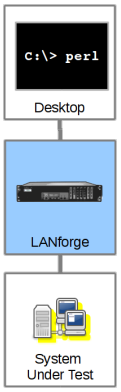
Multicast

Multicast endpoints are created differently both in the GUI and in the CLI scripting environment. This tutorial does not focus on multicast, but see the section Creating Endpoints section of Creating Connections with FIREMod Script and the chapter on WiFi Multicast Download.

Creating Connections with the FIREmod Script

Goal: Create, destroy, start and stop connections and endpoints without needing to use the LANforge GUI.

Traffic emulation can be run unattended and using automated tools without use of the LANforgeGUI using Perl scripts provided with the LANforge Server. These scripts can be run from within the LANforge server or outside the LANforge Server (on a Windows desktop). The output of the scripts needs to be redirected into a text file for you to process the results.



Script Capabilities

The `lf_firemod.pl` script has a lot of options because endpoints have a lot of features. Basic actions:

- Creating and Deleting Endpoints and Cross Connects: *create_endp, delte_endp, create_cx, delete_cx*
- Modifying an Endpoints TX Speed: *set_endp*
- Listing and Monitoring Ports, Endpoints and Cross Connects: *list_ports, show_endp, list_cx, show_cx*
- Reporting on Ports, Endpoints and Cross Connects: *show_port, show_endp, show_cx*
- Controlling Traffic: *do_cli, start_endp, stop_endp*. To start bi-directional traffic, start both endpoints.
- Pass direct CLI commands: *do_cmd*. Use this to help configure aspects of your testing scenario that are options presented in this script. Like secondary IPs on a port.

Creating a basic cross connect requires two endpoints, and each endpoint requires a port (network interface). Script options often begin by stating the manager, resource and action:

```
C:\> perl .\lf_firemod.pl --mgr 192.168.100.1 --resource 2 --action create_endp ...more options...
```

Script Actions, arguments to `--action`

Creating Endoints: `create_endp`

We use these parameters when creating and endpoint:

--endp_name
: name this endpoint

--port_name
: name of the port this endpoint uses

--speed
: speed of the endpoint transmission in bps

--tos
: type of service

--max_speed
: Maximum port speed if different than minimum speed of port, in bps

--endp_type
: Endpoint Types: *tcp, udp, tcp6, udp6*. To create multicast endpoint types, use *mc_udp* and *mc_udp6*.

--min_pkt_sz/--max_pkt_sz
: Minimum and maximum packet sizes

--use_csums
: Enable checksums

--ttl
: packet Time To Live

--report_timer
: the update interval for the endpoint

Example of creating a tcp connection endpoint with debugging:

Creating a multicast udp connection:

Create a connection with specific test-manager

Show Endpoint Stats: `show_endp`

By default, using the show_endp action shows all endpoints. It might be useful to place output like this right into a file or to immediate use grep to find the rows you want.

```
$ ./lf_firemod.pl --action show_endp --mgr cholla-f19

RSLT: 0  Cmd: 'nc_show_endp'


FileEndp [e2#0-nfs-100] [HOT_RUNNING, WRITING, WRITE_RATE_BURSTY, CHECK_MOUNT, AUTO_MOUNT, UN_MOUNT, O_TRUNC]
     Shelf: 1, Card: 1  Port: 10  Endpoint: 1  Type: FILE_NFS  Pattern: INCREASING
     MinWriteRate: 1544000bps  MaxWriteRate: 0bps  MinRead/WriteSz: 4096B  MaxRead/WriteSz: 32768B
     MinReadRate:  1544000bps  MaxReadRate:  1544000bps  QuiesceAfterFiles: -1
     NumFiles: 2  MinFileSize: 26214400B  MaxFileSize: 26214400B
     Directory: AUTO  Prefix: AUTO  Volume:
     Server-Mount: 10.41.0.1:/tank/tmp  Mount-Dir: AUTO  Mount-Options:
     RptTimer: 1000ms  RunningFor: 0s  StopIn: 0s  Quiesce: 3
     LastRpt: 0.000 secs ago     RealWriteRate: 0bps   RealReadRate: 0bps
     RetryTimer: 1000ms  InFailedIO: 0ms
       Buffers Read:     Total: 0          Time: 0s    Cur: 0         0/s
       Bytes Read:       Total: 0          Time: 0s    Cur: 0         0/s
       Files Read:       Total: 0          Time: 0s    Cur: 0         0/s
       Bytes Written:    Total: 0          Time: 0s    Cur: 0         0/s
       Buffers Written:  Total: 0          Time: 0s    Cur: 0         0/s
       Files Written:    Total: 0          Time: 0s    Cur: 0         0/s
       Read CRC Failed:  Total: 0          Time: 0s    Cur: 0         0/s

FileEndp [e2#0-nfs-101] [HOT_RUNNING, WRITING, WRITE_RATE_BURSTY, CHECK_MOUNT, AUTO_MOUNT, UN_MOUNT, O_TRUNC]
     Shelf: 1, Card: 1  Port: 12  Endpoint: 2  Type: FILE_NFS  Pattern: INCREASING
     MinWriteRate: 1544000bps  MaxWriteRate: 0bps  MinRead/WriteSz: 4096B  MaxRead/WriteSz: 32768B
     MinReadRate:  1544000bps  MaxReadRate:  1544000bps  QuiesceAfterFiles: -1
     NumFiles: 2  MinFileSize: 26214400B  MaxFileSize: 26214400B
     Directory: AUTO  Prefix: AUTO  Volume:
     Server-Mount: 10.41.0.1:/tank/tmp  Mount-Dir: AUTO  Mount-Options:
     RptTimer: 1000ms  RunningFor: 0s  StopIn: 0s  Quiesce: 3
     LastRpt: 0.000 secs ago     RealWriteRate: 0bps   RealReadRate: 0bps
     RetryTimer: 1000ms  InFailedIO: 0ms
       Buffers Read:     Total: 0          Time: 0s    Cur: 0         0/s
       Bytes Read:       Total: 0          Time: 0s    Cur: 0         0/s
       Files Read:       Total: 0          Time: 0s    Cur: 0         0/s
       Bytes Written:    Total: 0          Time: 0s    Cur: 0         0/s
       Buffers Written:  Total: 0          Time: 0s    Cur: 0         0/s
       Files Written:    Total: 0          Time: 0s    Cur: 0         0/s
       Read CRC Failed:  Total: 0          Time: 0s    Cur: 0         0/s
```

You can redirect all output into a file:

```
$ ./lf_firemod.pl --action show_endp --mgr cholla-f19 > /var/tmp/endp-stats.txt
```

It is possible to print out one-word attributes, such as MaxWriteRate tx_bps or rx_bps:

```
./lf_firemod.pl --mgr 127.0.0.1 --quiet 1 --action show_endp --endp_name cx_0-B --endp_vals tx_bps,rx_bps
Rx Bytes: 99938104
Tx Bytes: 99993112
```

Configure Endpoint: `set_endp`

This is for changing the attributes of an endpoint, such as endpoint TX rate.

```
$ ./lf_firemod.pl --mgr cholla-f19 --action set_endp --endp_name cx_0-A --speed 2000000
```

Show Port Stats: `show_port`

This is pretty useful for getting transmit rate on ports during a connection while not having to use the lf_portmod script. If you do not specify `--port_name`, all ports will be listed.

```
 $ ./lf_firemod.pl --action show_port --mgr cholla-f19 --port_name eth2#0

Shelf: 1, Card: 1, Port: 10  Type: MacVLAN  Alias:
  Win32-Name:   Win32-Desc:   Parent/Peer: eth2  Rpt-Timer: 8000  CPU-Mask: 0
   Current: UP LINK-UP TSO UFO GSO GRO PROBE_ERROR
   Supported: UP SEND_TO_SELF
   Partner: UP
   Advertising: 10bt-HD 10bt-FD 100bt-HD 100bt-FD 1000-FD TSO-ENABLED UFO-ENABLED GSO-ENABLED GRO-ENABLED
   IP: 10.41.0.10  MASK: 255.255.255.0  GW: 0.0.0.0  VID: 0  ResetState: COMPLETE
   DNS Servers:
   IPv6-Global: DELETED
   IPv6-Link: fe80::a00:27ff:fe09:183d/64
   IPv6-Gateway: DELETED
   MAC: 08:00:27:09:18:3d  DEV: eth2#0  MTU: 1500  TX Queue Len: 0
   LastDHCP: 0ms  Driver: macvlan Tx-Rate: 1000000Kbps
   Bus-Speed: 0/0  Bus-Width: 0/0
   Bridge-Port-Cost: Ignore  Prio: Ignore  Aging: 0
   DHCP-Client-ID: NONE  DHCP-Vendor-ID: NONE
     pps_tx: 0  pps_rx: 0  bps_tx: 0  bps_rx: 0
     Rxp: 5652  Txp: 21  Rxb: 1932984  Txb: 1826  RxERR: 0  TxERR: 0
     RxDrop: 0  TxDrop: 0  Multi: 5652  Coll: 0  RxLenERR: 0  RxOverFlow: 0
     RxCRC: 0  RxFrame: 0  RxFifo: 0  RxMissed: 0  TxAbort: 0  TxCarrier: 0
     TxFifo: 0  TxHeartBeat: 0  TxWindow: 0  RxBytesLL: 2068632  TxBytesLL: 2330
```

List Ports, action: `list_ports`

This is the same as `--show_port` without the port_name option.

Direct LANforge Command: `do_cmd`

In case you wanted to pass a CLI command directly in. Below is an example of setting the TOS flag for an endpoint:

```
C:\> perl .\lf_firemod.pl --mgr 192.168.100.1 --action do_cmd \
   --cmd "set_endp_tos cx_01-A LOWDELAY 10"
```

See the LANforge CLI User Guide for more info.

Remove endpoint: `delete_endp`

Remember to remove the cross connect before removing the endpoint.

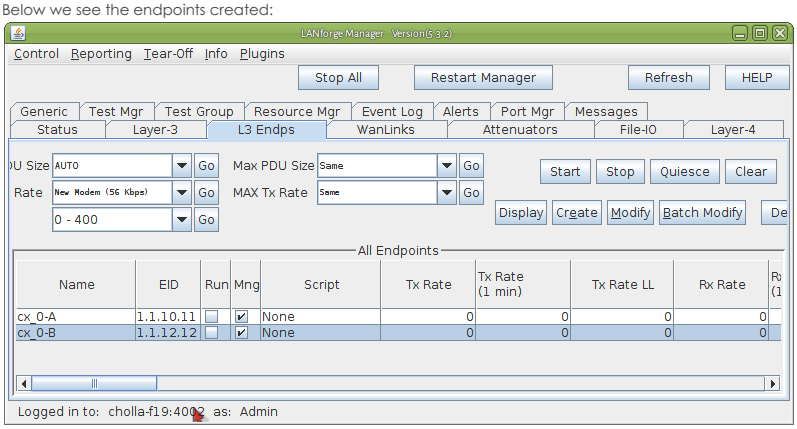
```
 $ ./lf_firemod.pl --action delete_endp  --mgr cholla-f19 --endp_name cx-0-A
```

Create Cross-connect: `create_cx`

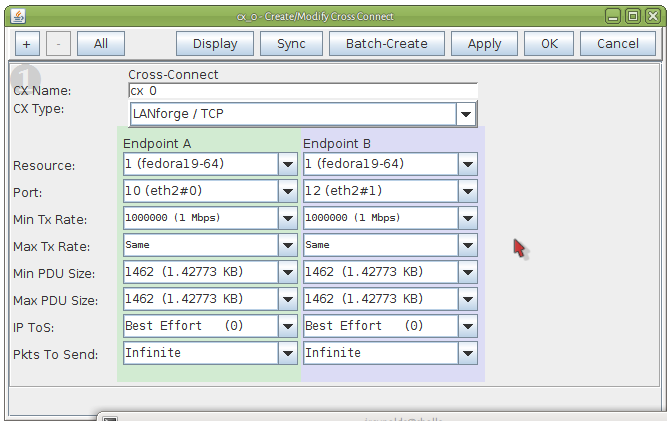
First you want to create two endpoints. You will add those endpoints to your cross connect. This example below shows all three steps:

```
$ ./lf_firemod.pl --action create_endp --mgr cholla-f19 --port_name eth2#0 \
     --endp_name cx_0-A --speed 1000000 --endp_type tcp --min_pkt_sz 1462 --report_timer 1000

$ ./lf_firemod.pl --action create_endp --mgr cholla-f19 --port_name eth2#1 \
     --endp_name cx_0-B --speed 1000000 --endp_type tcp --min_pkt_sz 1462 --report_timer 1000

$ ./lf_firemod.pl --action create_cx --mgr cholla-f19 --cx_name  cx_0 \
     --cx_endps cx_0-A,cx_0-B --report_timer 1000
```

Below we see the endpoints created:



and the CX details screen:



Show Cross Connects: `list_cx`

This shows the cross connects and their enpoints:

```
$ ./lf_firemod.pl --action list_cx --mgr cholla-f19
CX cx_0, endpoint cx_0-A, endpoint cx_0-B
```

Show Cross Connect Stats: `show_cx`

The output of this command begins with the basic stats for the CX and includes the statistics of each endpoint.

```
$ ./lf_firemod.pl --action show_endp  --mgr cholla-f19 --endp_name cx_0-A
```

```
 LANFORGE_TCP CX:  cx_0  id: 12  type: LANFORGE_TCP  DesiredState: UN_INITIALIZED RealState: STOPPED
   tx_endpoint: cx_0-A  rx_endpoint: cx_0-B report_timer: 1.000000s  TestMgr: default_tm

Endpoint [cx_0-A] (NOT_RUNNING, FIXED_PLD_SIZE, RATE_BURSTY, IP_PORT_AUTO)
     Shelf: 1, Card: 1  Port: 10  Endpoint: 11 Type: LANFORGE_TCP  Pattern: INCREASING
     MinTxRate: 1000000bps  MaxTxRate: 1000000bps  MinPktSize: 1462B  MaxPktSize: 1462B
     DestMAC: 08:00:27:69:1a:3d  DestIpAddr: 10.41.0.11  DestIpPort: 0  Quiesce: 3
     SrcMAC:  08:00:27:09:18:3d  SrcIp:  0.0.0.0  IpPort:  0-0  IpTOS: DONT-SET  Priority: 0
     Role: CONNECT  RptTimer: 1000ms  RunningFor: 0s  StopIn: 0s  Avg-Jitter: 0ms
     Latency: 0 -:0:- 0  [ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ] (1)
     Pkt-Gaps: 0 -:0:- 0  [ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ] (1)
     Last-Rpt: 0.000 secs ago  RealTxRate: 0bps  RealRxRate: 0bps  TTL: 0  Conn-Timeout: 10000
     FileName:  SendBadCrc: 0  RcvBuf: 0  SndBuf: 0  CWND: 0 SND-MSS: 0
     RxDrop%-SEQ:  0.0000  RxDrop%-CX: 0.0000  Conn-Timer: -1-0ms  Conn-Pause: 0-0ms
     PktsToSend: 0
  Multi-Conn: 0  Active-Connections: 0  Files-Played: 0
     RunningInGroup: NONE  Script-steps-completed: 0  Steps-Failed: 0
     First-Rx: -1ms  Mcast-Source: 0.0.0.0:0
       Rx Pkts:            Total: 0            Time: 60s   Cur: 0        0/s
       Rx Pkts (On Wire):  Total: 0            Time: 60s   Cur: 0        0/s
       Rx Bytes:           Total: 0            Time: 60s   Cur: 0        0/s
       Rx Bytes (On Wire): Total: 0            Time: 60s   Cur: 0        0/s
       Rx OOO Pkts:        Total: 0            Time: 60s   Cur: 0        0/s
       RX Wrong Dev:       Total: 0            Time: 60s   Cur: 0        0/s
       RX CRC Failed:      Total: 0            Time: 60s   Cur: 0        0/s
       RX Bit Errors:      Total: 0            Time: 3s    Cur: 0        0/s
       Rx Dropped Pkts:    Total: 0            Time: 3s    Cur: 0        0/s
         Cx Detected:      0
       Rx Duplicate Pkts:  Total: 0            Time: 60s   Cur: 0        0/s
       Tx Pkts:            Total: 0            Time: 60s   Cur: 0        0/s
       Tx Pkts (On Wire):  Total: 0            Time: 60s   Cur: 0        0/s
       Tx Bytes:           Total: 0            Time: 60s   Cur: 0        0/s
       Tx Bytes (On Wire): Total: 0            Time: 3s    Cur: 0        0/s
       Tx Failed Pkts:     Total: 0            Time: 60s   Cur: 0        0/s
       Tx Failed Bytes:    Total: 0            Time: 60s   Cur: 0        0/s
       Conn Established:   Total: 0            Time: 30s   Cur: 0        0/s
       TCP Retransmits:    Total: 0            Time: 3s    Cur: 0        0/s
       Conn Timeouts:      Total: 0            Time: 30s   Cur: 0        0/s

Endpoint [cx_0-B] (NOT_RUNNING, FIXED_PLD_SIZE, RATE_BURSTY, IP_PORT_AUTO)
     Shelf: 1, Card: 1  Port: 12  Endpoint: 12 Type: LANFORGE_TCP  Pattern: INCREASING
     MinTxRate: 1000000bps  MaxTxRate: 1000000bps  MinPktSize: 1462B  MaxPktSize: 1462B
     DestMAC: 08:00:27:09:18:3d  DestIpAddr: 10.41.0.10  DestIpPort: 0  Quiesce: 3
     SrcMAC:  08:00:27:69:1a:3d  SrcIp:  0.0.0.0  IpPort:  0-0  IpTOS: DONT-SET  Priority: 0
     Role: ACCEPT  RptTimer: 1000ms  RunningFor: 0s  StopIn: 0s  Avg-Jitter: 0ms
     Latency: 0 -:0:- 0  [ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ] (1)
     Pkt-Gaps: 0 -:0:- 0  [ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ] (1)
     Last-Rpt: 0.000 secs ago  RealTxRate: 0bps  RealRxRate: 0bps  TTL: 0  Conn-Timeout: 10000
     FileName:  SendBadCrc: 0  RcvBuf: 0  SndBuf: 0  CWND: 0 SND-MSS: 0
     RxDrop%-SEQ:  0.0000  RxDrop%-CX: 0.0000  Conn-Timer: -1-0ms  Conn-Pause: 0-0ms
     PktsToSend: 0
  Multi-Conn: 0  Active-Connections: 0  Files-Played: 0
     RunningInGroup: NONE  Script-steps-completed: 0  Steps-Failed: 0
     First-Rx: -1ms  Mcast-Source: 0.0.0.0:0
       Rx Pkts:            Total: 0            Time: 60s   Cur: 0        0/s
       Rx Pkts (On Wire):  Total: 0            Time: 60s   Cur: 0        0/s
       Rx Bytes:           Total: 0            Time: 60s   Cur: 0        0/s
       Rx Bytes (On Wire): Total: 0            Time: 60s   Cur: 0        0/s
       Rx OOO Pkts:        Total: 0            Time: 60s   Cur: 0        0/s
       RX Wrong Dev:       Total: 0            Time: 60s   Cur: 0        0/s
       RX CRC Failed:      Total: 0            Time: 60s   Cur: 0        0/s
       RX Bit Errors:      Total: 0            Time: 3s    Cur: 0        0/s
       Rx Dropped Pkts:    Total: 0            Time: 3s    Cur: 0        0/s
         Cx Detected:      0
       Rx Duplicate Pkts:  Total: 0            Time: 60s   Cur: 0        0/s
       Tx Pkts:            Total: 0            Time: 60s   Cur: 0        0/s
       Tx Pkts (On Wire):  Total: 0            Time: 60s   Cur: 0        0/s
       Tx Bytes:           Total: 0            Time: 60s   Cur: 0        0/s
       Tx Bytes (On Wire): Total: 0            Time: 3s    Cur: 0        0/s
       Tx Failed Pkts:     Total: 0            Time: 60s   Cur: 0        0/s
       Tx Failed Bytes:    Total: 0            Time: 60s   Cur: 0        0/s
       Conn Established:   Total: 0            Time: 30s   Cur: 0        0/s
       TCP Retransmits:    Total: 0            Time: 3s    Cur: 0        0/s
       Conn Timeouts:      Total: 0            Time: 30s   Cur: 0        0/s
```

Remove Cross Connect: `delete_cx`

Remember to delete a cross connect before you delete its endpoints.

```
$ ./lf_firemod.pl --action delete_cx  --mgr cholla-f19 --cx_name cx_0
```

Controlling Traffic

You need to use `do_cmd` to control Unicast traffic.

By default, cross connects are created in the *default_tm* test manager. To control them, you want to specify *default_tm* in your *set_cx_state* CLI command.

```
./lf_firemod.pl --mgr 127.0.0.1 --quiet 0 --action do_cmd --cmd "set_cx_state default_tm cx_0 RUNNING"
```

The format of the command is specified in the CLI User Guide: set_cx_state. Possible CX states include:

- RUNNING
- SWITCH
- QUIECE
- STOPPED
- DELETED

For Multicast traffic, use `start_endp/stop_endp`

```
$ ./lf_firemod.pl --mgr cholla-f19 --action stop_endp --endp_name cx_0-A
```

Multicast Endpoints

There are different options for creating multicast endpoints.

```
$ ./lf_firemod.pl --action create_endp --endp_name mcast_xmit_1 \
  --endp_type mc_udp      --speed 154000 \
  --mcast_addr 224.9.9.8  --mcast_port 9998 \
  --rcv_mcast NO          --port_name eth1 \
  --min_pkt_sz 1072       --max_pkt_sz 1472 \
  --use_csums NO          --ttl 32
```

Add secondary IPs to a Port

This is not a default script option, so we use the `do_cmd` action:

```
C:\> perl .\lf_firemod.pl --mgr 192.168.100.1
  --action do_cmd "set_sec_ip 1 1 eth1 10.26.0.20-250/24"
```

See the LANforge CLI User Guide for more info.

Present Options

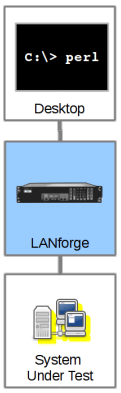
This is the output of `lf_firemod.pl --help`:

```
./lf_firemod.pl  --action { create_endp | show_endp | set_endp | show_port | list_ports
                          | do_cmd | start_endp | stop_endp | delete_endp
                          | create_cx | list_cx | show_cx | delete_cx } ]
  [--endp_vals {key,key,key,key}]
      # show_endp output can be narrowed with key-value arguments
      # Examples:
      #  --action show_endp --endp_vals MinTxRate,DestMAC,Avg-Jitter
      # Not available: Latency,Pkt-Gaps, or rows below steps-failed.
      # Special Keys:
      #  --endp_vals tx_bps         (Tx Bytes)
      #  --endp_vals rx_bps         (Rx Bytes)
  [--mgr      {host-name | IP}]
  [--mgr_port  {ip port}]
  [--cmd      {lf-cli-command text}]
  [--endp_name {name}]
  [--port_name {name}]
  [--resource  {number}]
  [--speed     {speed in bps}]
  [--tos       { DONT-SET | LOWDELAY | THROUGHPUT | RELIABILITY | LOWCOST },{priority}]
  [--max_speed {speed in bps}]
  [--quiet     { yes | no }]
  [--endp_type { lf_udp | lf_udp6 | lf_tcp | lf_tcp6 | mc_udp | mc_udp6 }]
  [--mcast_addr {multicast address, for example: 224.4.5.6}]
  [--mcast_port {multicast port number}]
  [--min_pkt_sz {minimum payload size in bytes}]
  [--max_pkt_sz {maximum payload size in bytes}]
  [--rcv_mcast { yes (receiver) | no (transmitter) }]
  [--use_csums { yes | no, should we checksum the payload }]
  [--ttl       {time-to-live}]
  [--report_timer {milliseconds}]
  [--cx_name   {connection name}]
  [--cx_endps  {endp1},{endp2}]
  [--test_mgr  {default_tm|all|other-tm-name}]

Example:
 ./lf_firemod.pl --action set_endp --endp_name udp1-A --speed 154000

 ./lf_firemod.pl --action create_endp --endp_name mcast_xmit_1 --speed 154000 \
   --endp_type mc_udp   --mcast_addr 224.9.9.8 --mcast_port 9998 \
   --rcv_mcast NO       --port_name eth1 \
   --min_pkt_sz 1072    --max_pkt_sz 1472 \
   --use_csums NO       --ttl 32 \
   --quiet no --report_timer 1000

 ./lf_firemod.pl --action create_endp --endp_name bc1 --speed 256000 \
   --endp_type lf_tcp   --tos THROUGHPUT,100 --port_name rd0#1

 ./lf_firemod.pl --action list_cx --test_mgr all --cx_name all

 ./lf_firemod.pl --action create_cx --cx_name L301 \
   --cx_endps ep_rd0a,ep_rd1a --report_timer 1000
```

Creating Endpoint Hunt Scripts with CLI API

Goal: Use the the CLI to operate the Endpoint Scripting features of the Layer-3 Endpoints you create.

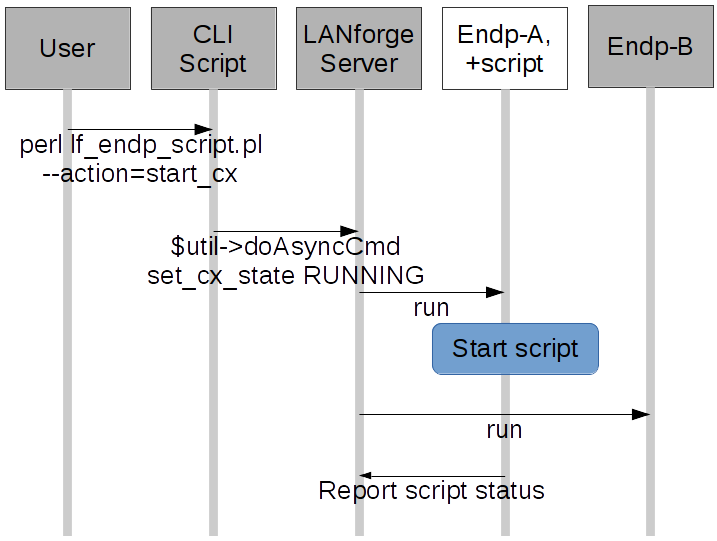
Layer-3 endpoints can manipulate their own transmission parameters using a variety of internal scripts, known as *Endpoint Scripts*. Using the `lf_endp_script.pl` CLI script, you can operate those internal endpoints behaviours.



> *This cookbook talks about Endpoint Scripts and CLI scripts at the same time. In this chapter, if the term **script** is used, assume **Endpoint Script**.* Additionally, the terms operating and running can also be confusing. To keep the activities distinct, a LANforge *user* will **operate** a CLI script from a terminal. The LANforge *server* will **run** the Endpoint Script. A **CLI script** is a user-space perl script that issues CLI commands to a LANforge server. A **CLI command** is an instruction obeyed by the LANforge server.

The Forces at Play

There are a number of subsystems running while we operate an automated Endpoint Script, so let's review them:

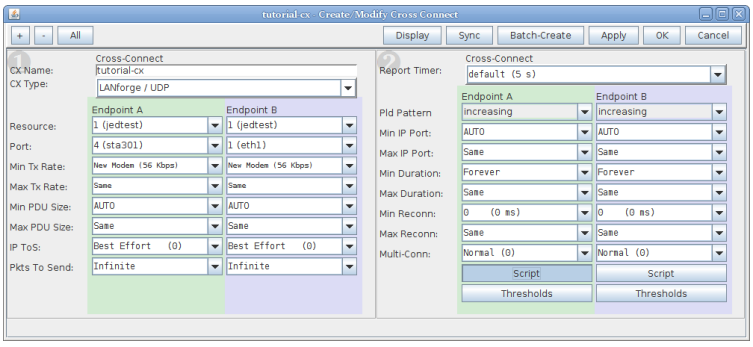


- There will be Layer-3 connect constructed using `lf_firemod.pl`. (Don't forget: create the endpoints before creating the cross connect.)
- A *managed endpoint* of that connection will be configured with an Endpoint Script.
- The attending engineer will operate a CLI script that changes state the Layer-3 connection to Running
- The Layer-3 connection starts both endpoints transmitting, one of them starts running it's Endpoint Script that sets it's transmit parameters.
- Remember: Endpoint Scripts run inside the LANforge server process. CLI scripts run from the client side.

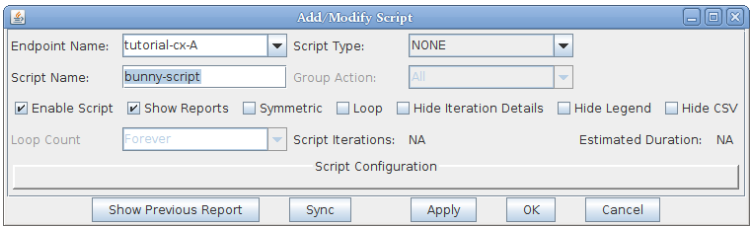
Let's Walk Thru Putting One Together

We recommend starting your first script off by the LANforge GUI to save an endpoint with an Endpoint Script. Next, inspect the LANforge database on the server for the script parameters. Take those parameters and adapt them to the operator's CLI script.

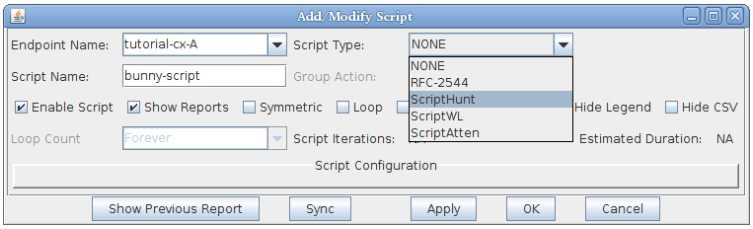
1. From the *Layer-3* tab, open a connection **tutorial-cx**, and navigate to **box 2**. Click on the **Script** button.



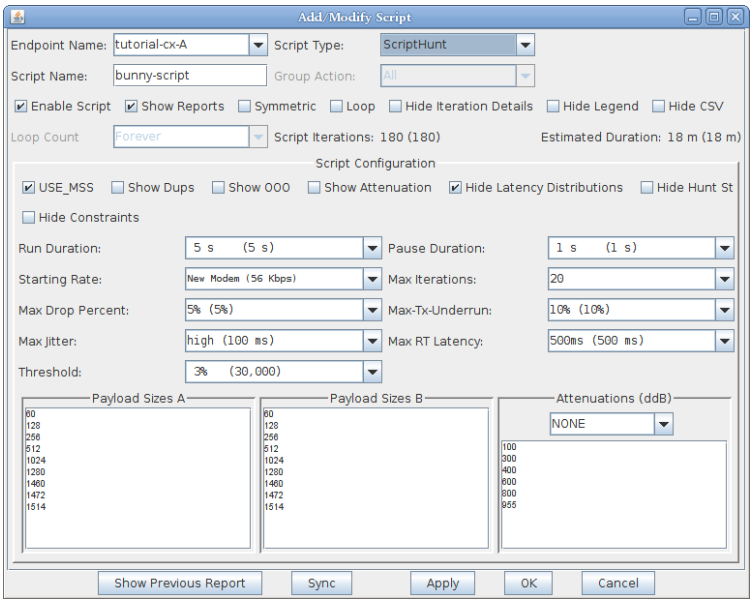
2. Name your script



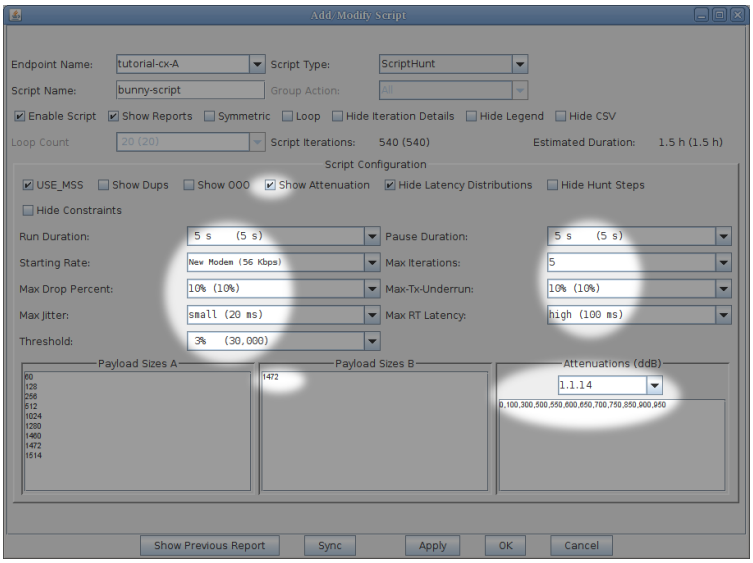
3. Select your Script type, here we choose **ScriptHunt**



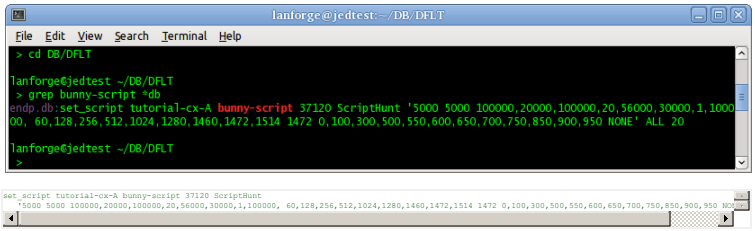
4. We immediately see the parameters for the script:



5. Let's modify the parameters to match our CLI command example below:



6. In a LANforge terminal, let's look at at `/home/lanforge/DB/DFLT/endps.db` We will search for `bunny-script` and we'll inspect the resulting CLI command.



7. Now we can craft this command into a CLI script. In a CMD window, we can write the formatted CLI script arguments:

```
C:\> .\lf_endp_script.pl --mgr jedtest --resource 1 ^
 --action set_script --script_type Hunt --script_name bunny-script ^
 --endp_name tutorial-CX-A -loops 1 --flags 37120 ^
 --private "5000 5000 100000,20000,100000,20,56000,30000,1,100000, 60,128,256,512,1024,1280,1460,1472,1514 1472 0,100,300
```

> i In the CMD window, use double-quotes `"` for quoted script arguments. Using single-quotes will break your command.

In a Linux terminal, we can use double `"` or single `'` quotes:

```
$ ./lf_endp_script.pl --mgr jedtest --resource 1 \
 --action set_script --script_type Hunt --script_name bunny-script \
 --endp_name tutorial-CX-A -loops 1 --flags 37120 \
 --private '5000 5000 100000,20000,100000,20,56000,30000,1,100000, 60,128,256,512,1024,1280,1460,1472,1514 1472 0,100,300
```

8. We can start the connection and the Endpoint Script will immediately begin running:

```
lf_endp_script --mgr jedtest --resource 1 --action start_cx --cx_name tutorial-CX
```

9. If the number of loops is fixed, it will eventually quiesce and stop itself. If we need to stop it and let in-flight packets come to rest, we can quiesce it:

```
lf_endp_script --mgr jedtest --resource 1 --action quiesce_cx --cx_name tutorial-CX
```

We could also use action `stop_cx` to immediately stop the connection.

10. If you have a LANforge GUI running, the Endpoint Script report will automatically display in a GUI window as soon as the connection starts. To display it to the terminal, you need to enable debug output:

```
lf_endp_script.pl --action show_report --endp_name tutorial-CX-A --quiet no
```

Or to save it to a text file:

```
lf_endp_script.pl --action show_report --endp_name tutorial-CX-A --quiet no > /home/lanforge/Documents/report.txt
```

11. To remove the script:

```
lf_endp_script.pl --action remove_script --endp_name tutorial-CX-A
```

At the CLI Command Level

Review of the `set_script` CLI command

We have covered creating endpoints in earlier cookbooks. The perl script `lf_endp_script.pl` was created to modify endpoints and operate their Endpoint Scripts. That script is using the `set_script` CLI command (documented here). A call to it looks like:

```
set_script tutorial-cx-A bunny-script 37120 ScriptHunt '...' ALL 20
```

Endpoint Scripting Uses Large Parameters

That vague `'...'` section is the **private** parameter which is a parameter list each script type requires. The private parameter combines a series of constraints (sub-parameters). For the ScriptHunt, we might use:

```
run_duration pause_duration constraints payload_sizes_a payload_sizes_b attenuations attenuator
 5000          |            |                |                  |              |          |
            5000            |                |                  |              |          |
100000,20000,100000,20,56000,30000,1,100000  |                  |              |          |
                  | 60,128,256,512,1024,1280,1460,1472,1514     |              |          |
                  |                                            1472           |          |
                  |                     0,100,300,500,550,600,650,700,750,850,900,950      |
                  v                                                                      1.1.14
      drops,jitter_us,latency_us,max_steps,start_rate,accuracy,is_bps,max_tx_slowdown
```

Accuracy is also Threshold, max_tx_slowdown is also Underrun. The result is a very long line that has to be surrounded the the CLI level by one pair of single quotes:

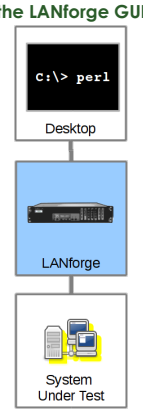
```
'5000 5000 100000,20000,100000,20,56000,30000,1,100000, 60,128,256,512,1024,1280,1460,1472,1514 1472 0,100,300,500,550,600,650,700,750,850,900,950 NONE'
```

> i Write these parameters very carefully! Your first mistake is likely going to involve misplaced apostrophes.

Associating stations with the lf_associate_ap script.

Goal: Create, destroy, start and stop virtual stations without needing to use the LANforge GUI.

Automated wireless traffic is possible using the `lf_associate_ap.pl` script. This script can be run within the LANforge server or outside the LANforge Server (on a windows desktop). The output of the script should be redirected to a text file if you want to review the resuts. Use this file in conjunction with the `lf_firemod.pl` script to create traffic. Requires a LANforge CT520 (or better) system and an access point.



Script Capabilities

The `lf_assocatiate_ap.pl` script has many options, but here are the basic actions:

- Create stations and cross connects with them, running traffic for a specified amount of time *(action: step1)*
- Generate stress on the AP by repeatedly bringing up stations and taking them down *(action: step2)*.

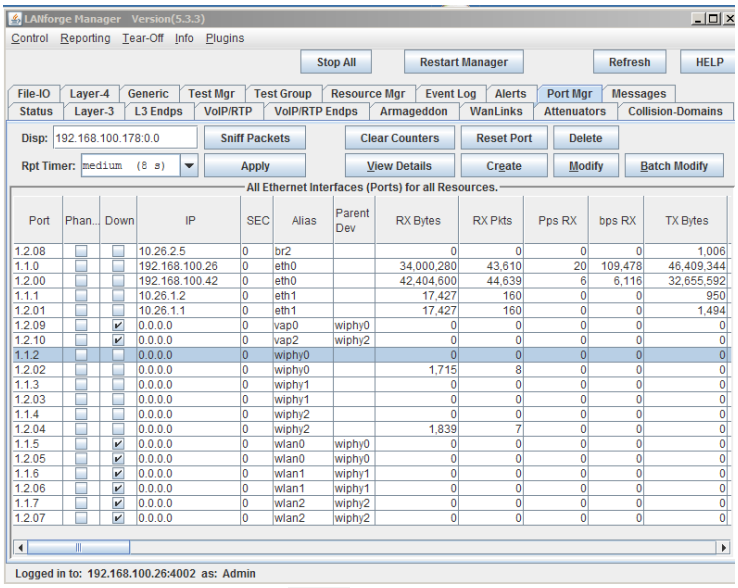
Before you begin

1. We assume you have a separate WiFi access point in routed mode. These examples can be used on a CT523 (or better) system with more than one radio if you want to practice the techniques. You would dedicate a radio to be a virtual AP (see cookbook).
2. For these examples, our AP will be open with no username or password, and the SSID will be **jedtest**
3. If you want to run scripts from your Windows desktop, you have ActivePerl installed.

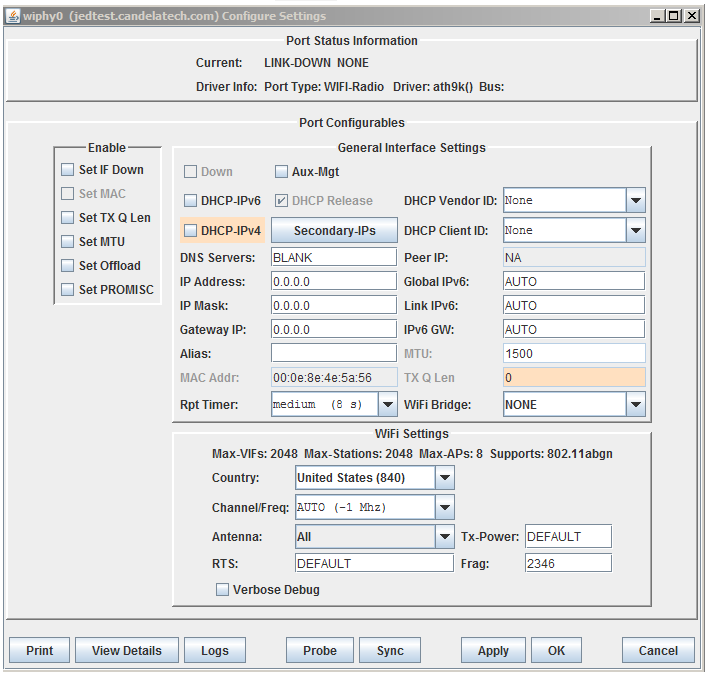
Creating a virtual station with traffic

Using `lf_associate_ap` on Windows

1. In the LANforge GUI, we will inspect our `wiphy0` radio.



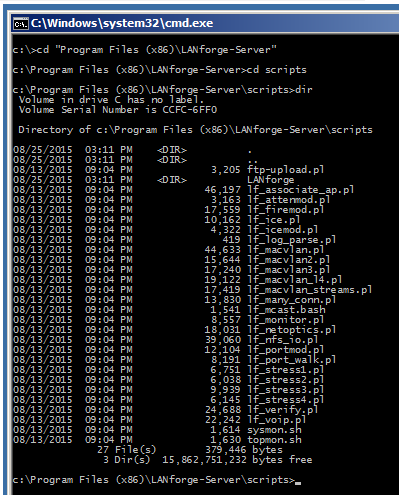
And the radio should be set to channel `-1 AUTO`



CMD window shortcut: ⊞ R cmd ↵

LANforge Scripts are at **`C:\Program Files\LANforge-Server\scripts`**

2. `cd C:\Program Files\LANforge-Server\scripts`



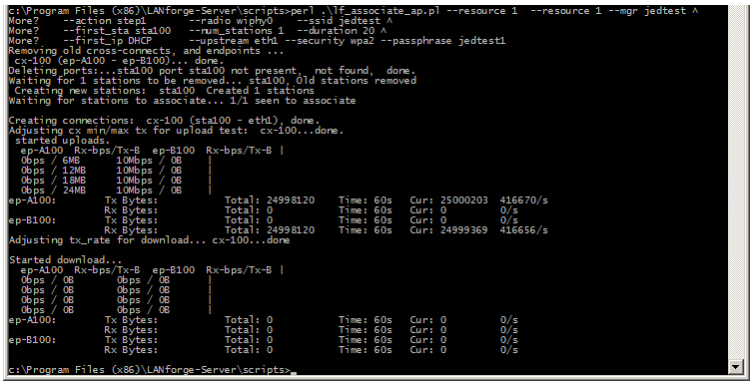
3. `perl .\lf_associate_ap.pl --help` Will show you the script options.

```
C:\Windows\system32\cmd.exe

c:\Program Files (x86)\LANforge-Server\scripts>perl .\lf_associate_ap.pl --help
Unknown option: help
.\lf_associate_ap.pl   [--mgr {host-name | IP}]
      [--mgr_port {ip port}]      # use if on non-default management port
      [--resource {resource}]     # use if multiple lanforge systems; defaults to 1
      [--quiet { yes | no }]      # debug output; -q

      ##       AP selection
      [--radio {name}]            # e.g. wiphy2
      [--ssid {ssid}]             # e.g. jedtest
      [--security {open|wep|wpa|wpa2}] # station authentication type
      [--passphrase {...}]          # implies wpa2 if --security not set
      [--wifi_mode {a|abg|abgn|abgnAC|an|anAC|b|bg|bgn|g}]

      ##       station configuration
      [--num_stations {10}]
      [--first_sta {sta100}]
      [--first_ip {DHCP |ip address}]
      [--netmask {255.255.0.0}]

      ##       connection configuration
      [--cxtype {tcp/tcp6/udp/udp6}]   # use a tcp/udp connection, default tcp
      [--upstream {name|eth1}]
         # could be AP or could be port on LANforge
         # connected to WAN side of AP
      [--bps-min {1000000000}]      # minimum tx bps
      [--bps-max {SAME|bps-value}]  # maximum tx bps, use SAME or omit for SAME
      [--duration {30}]      # connection duration, seconds, default 60
      [--poll-time {5}]    # nap time between connection displays
      [--action {step1,step2}]
         # step1: creates <num_stations> stations and L3 connections
         # step2: does bringup test

      [--traffic_type {separate|concurrent}]
         # for step1: separate does download then upload
         # concurrent does upload and download at same time

      [--db_preload {scenario name}]
         # load this database before creating stations
         # option intended as a cleanup step

      [--db_save {name}]
         # save the state of this test scenario after running the
         # connections, before --db_postload

      [--db_postload {scenario name}]
         # load this database after running connections,
         # option intended as a cleanup step

Examples:
## connecting to an open AP, at 2Mbps, for 20 minutes
 .\lf_associate_ap.pl --action step1   --radio wiphy0  --ssid ap-test-01  \
   --bps-min 2000000 --duration 1200 --upstream eth1

 .\lf_associate_ap.pl --action step2 --sta_names tcp-sta1,tcp-sta2,tcp-sta3 --ssid ap-test-01

## using a second lanforge system to connect to wpa2 AP:
 .\lf_associate_ap.pl --mgr 192.168.100.1 --resource 2 --radio wiphy2 \
   --ssid jedtest       --passphrase 'asdf1234' \
   --num_stations 10    --first_sta sta400 \
   --first_ip DHCP      --upstream eth1 --action step1

## (Windows) using a beginning database and saving the resulting database:
 C:\Users\bob>cd "C:\Program Files (x86)\LANforge-Server\scripts"
 C:\Program Files (x86)\LANforge-Server\scripts>perl lf_associate_ap.pl --mgr jedtest \
 --resource 2  --radio wiphy2 --first_ip DHCP \
 --duration 10 --bps-min 10k  --bps-max 20M --cxtype tcp \
 --ssid jedtest --passphrase jedtest1 --security wpa2 \
 --first_sta 300 --db_preload Radio2 --db_save run_results --num_stations 3

## connecting to wpa AP:
.\lf_associate_ap.pl --mgr 192.168.100.1 --radio wiphy0 \
   --ssid jedtest    --passphrase 'asdf1234' --security wep \
   --num_stations 10 --first_sta sta400 \
   --first_ip DHCP   --upstream eth1       --action step1


c:\Program Files (x86)\LANforge-Server\scripts>
```

4. We can create a virtual station with this command:

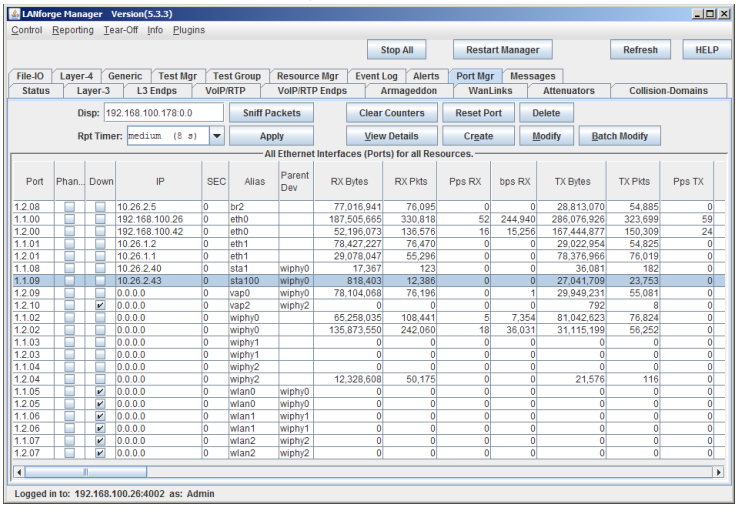
```
perl .\lf_associate_ap.pl --resource 1  --resource 1 --mgr jedtest ^
   --action step1        --radio wiphy0    --ssid jedtest ^
   --first_sta sta100    --num_stations 1  --duration 20 ^
   --first_ip DHCP       --upstream eth1 --security wpa2 --passphrase jedtest1
```

> i Long DOS commands and be continued on the next line with the ^ character.



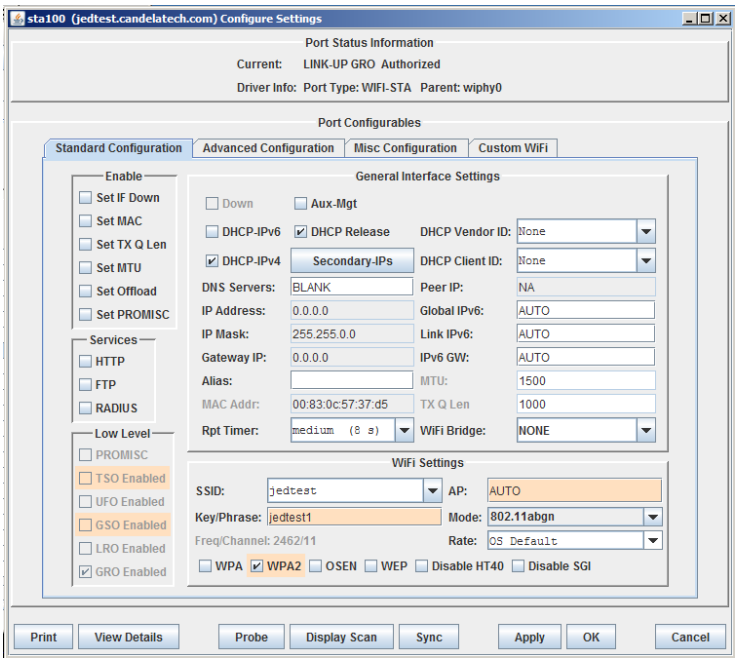
```
c:\Program Files (x86)\LANforge-Server\scripts>perl .\lf_associate_ap.pl --resource 1  --resource 1 --mgr jedtest ^
More?    --action step1       --radio wiphy0    --ssid jedtest ^
More?    --first_sta sta100   --num_stations 1  --duration 20 ^
More?    --first_ip DHCP      --upstream eth1 --security wpa2 --passphrase jedtest1
Removing old cross-connects, and endpoints ...
 cx-100 (ep-A100 - ep-B100)... done.
Deleting ports:...sta100 port sta100 not present,  not found,  done.
Waiting for 1 stations to be removed... sta100, Old stations removed
 Creating new stations:  sta100  Created 1 stations
Waiting for stations to associate... 1/1 seen to associate

Creating connections:  cx-100 (sta100 - eth1), done.
Adjusting cx min/max tx for upload test:  cx-100...done.
 started uploads.
  ep-A100  Rx-bps/Tx-B  ep-B100  Rx-bps/Tx-B |
  0bps / 6MB     10Mbps / 0B     |
  0bps / 12MB    10Mbps / 0B     |
  0bps / 18MB    10Mbps / 0B     |
  0bps / 24MB    10Mbps / 0B     |
ep-A100:        Tx Bytes:          Total: 24998120    Time: 60s   Cur: 25000203  416670/s
                Rx Bytes:          Total: 0           Time: 60s   Cur: 0         0/s
ep-B100:        Tx Bytes:          Total: 0           Time: 60s   Cur: 0         0/s
                Rx Bytes:          Total: 24998120    Time: 60s   Cur: 24999369  416656/s
Adjusting tx_rate for download... cx-100...done

Started download...
  ep-A100  Rx-bps/Tx-B  ep-B100  Rx-bps/Tx-B |
  0bps / 0B      0bps / 0B      |
  0bps / 0B      0bps / 0B      |
  0bps / 0B      0bps / 0B      |
  0bps / 0B      0bps / 0B      |
ep-A100:        Tx Bytes:          Total: 0           Time: 60s   Cur: 0         0/s
                Rx Bytes:          Total: 0           Time: 60s   Cur: 0         0/s
ep-B100:        Tx Bytes:          Total: 0           Time: 60s   Cur: 0         0/s
                Rx Bytes:          Total: 0           Time: 60s   Cur: 0         0/s

c:\Program Files (x86)\LANforge-Server\scripts>
```

5. We can see the port appear in the LANforge GUI:



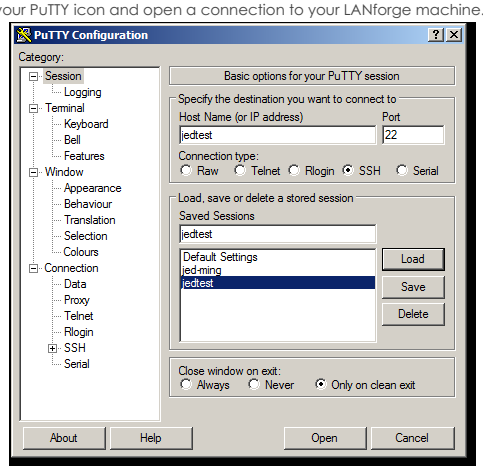
Port	Phan...	Down	IP	SEC	Alias	Parent Dev	RX Bytes	RX Pkts	Pps RX	bps RX	TX Bytes	TX Pkts	Pps TX
1.2.08			10.26.2.5	0	br2		77,016,941	76,095	0	0	28,813,070	54,885	0
1.1.00			192.168.100.26	0	eth0		187,505,665	330,818	52	244,940	286,076,926	323,699	59
1.2.00			192.168.100.42	0	eth0		52,196,073	136,576	16	15,256	167,444,877	150,309	24
1.1.01			10.26.1.2	0	eth1		78,427,227	76,470	0	0	29,022,954	54,825	0
1.2.01			10.26.1.1	0	eth1		29,078,047	55,296	0	0	78,376,966	76,019	0
1.1.08			10.26.2.40	0	sta1	wiphy0	17,367	123	0	0	36,081	182	0
1.1.09			10.26.2.43	0	sta100	wiphy0	818,403	12,386	0	0	27,041,709	23,753	0
1.2.09			0.0.0.0	0	vap0	wiphy0	78,104,068	76,196	0	1	29,949,231	55,081	0
1.2.10		✔	0.0.0.0	0	vap2	wiphy2	0	0	0	0	792	8	0
1.1.02			0.0.0.0	0	wiphy0		65,258,035	108,441	5	7,354	81,042,623	76,824	0
1.2.02			0.0.0.0	0	wiphy0		135,873,550	242,060	18	36,031	31,115,199	56,252	0
1.1.03			0.0.0.0	0	wiphy1		0	0	0	0	0	0	0
1.2.03			0.0.0.0	0	wiphy1		0	0	0	0	0	0	0
1.1.04			0.0.0.0	0	wiphy2		0	0	0	0	0	0	0
1.2.04			0.0.0.0	0	wiphy2		12,328,608	50,175	0	0	21,576	116	0
1.1.05		✔	0.0.0.0	0	wlan0	wiphy0	0	0	0	0	0	0	0
1.2.05		✔	0.0.0.0	0	wlan0	wiphy0	0	0	0	0	0	0	0
1.1.06		✔	0.0.0.0	0	wlan1	wiphy1	0	0	0	0	0	0	0
1.2.06		✔	0.0.0.0	0	wlan1	wiphy1	0	0	0	0	0	0	0
1.1.07		✔	0.0.0.0	0	wlan2	wiphy2	0	0	0	0	0	0	0
1.2.07		✔	0.0.0.0	0	wlan2	wiphy2	0	0	0	0	0	0	0

and we can inspect it.

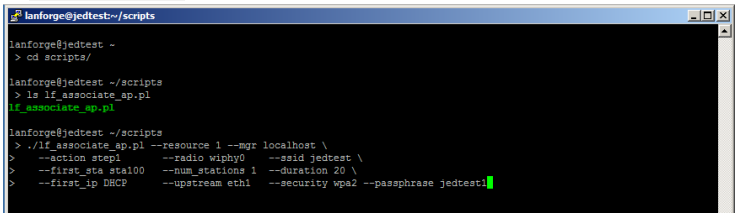


Using `lf_associate_ap` on Linux

1. Double click on your PuTTY icon and open a connection to your LANforge machine.



2. The `lf_associate_ap.pl` script is in the *scripts* sub directory.

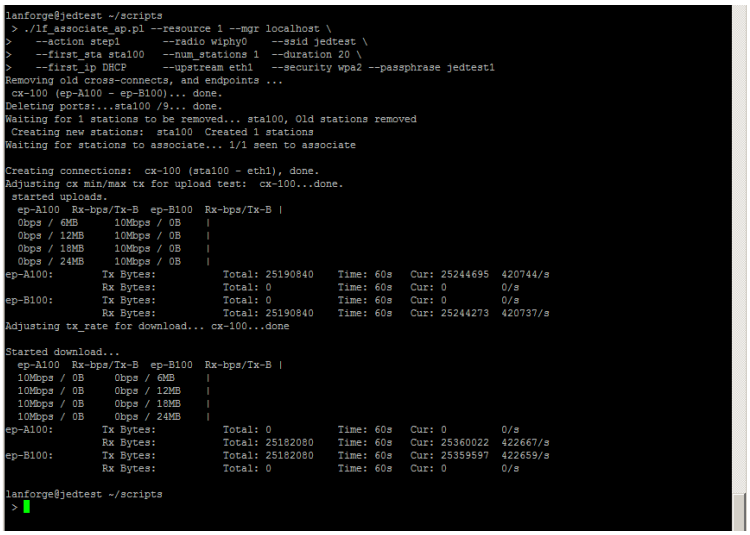


3. Our command is basically the same.

> i Long shell commands and be continued on the next line with the `\` character.

```
./lf_associate_ap.pl --resource 1 --mgr localhost \
   --action step1        --radio wiphy0    --ssid jedtest \
   --first_sta sta100    --num_stations 1  --duration 20 \
   --first_ip DHCP       --upstream eth1   --security wpa2 --passphrase jedtest1
```

4. We will see similar output:



```
lanforge@jedtest ~/scripts
 > ./lf_associate_ap.pl --resource 1 --mgr localhost \
>    --action step1       --radio wiphy0    --ssid jedtest \
>    --first_sta sta100   --num_stations 1  --duration 20 \
>    --first_ip DHCP      --upstream eth1   --security wpa2 --passphrase jedtest1
Removing old cross-connects, and endpoints ...
 cx-100 (ep-A100 - ep-B100)... done.
Deleting ports:...sta100 /9... done.
Waiting for 1 stations to be removed... sta100, Old stations removed
 Creating new stations:  sta100  Created 1 stations
Waiting for stations to associate... 1/1 seen to associate

Creating connections:  cx-100 (sta100 - eth1), done.
Adjusting cx min/max tx for upload test:  cx-100...done.
 started uploads.
  ep-A100  Rx-bps/Tx-B  ep-B100  Rx-bps/Tx-B |
  0bps / 6MB      10Mbps / 0B    |
  0bps / 12MB     10Mbps / 0B    |
  0bps / 18MB     10Mbps / 0B    |
  0bps / 24MB     10Mbps / 0B    |
ep-A100:        Tx Bytes:          Total: 25190840    Time: 60s   Cur: 25244695  420744/s
                Rx Bytes:          Total: 0           Time: 60s   Cur: 0         0/s
ep-B100:        Tx Bytes:          Total: 0           Time: 60s   Cur: 0         0/s
                Rx Bytes:          Total: 25190840    Time: 60s   Cur: 25244273  420737/s
Adjusting tx_rate for download... cx-100...done

Started download...
  ep-A100  Rx-bps/Tx-B  ep-B100  Rx-bps/Tx-B |
  10Mbps / 0B     0bps / 6MB     |
  10Mbps / 0B     0bps / 12MB    |
  10Mbps / 0B     0bps / 18MB    |
  10Mbps / 0B     0bps / 24MB    |
ep-A100:        Tx Bytes:          Total: 0           Time: 60s   Cur: 0         0/s
                Rx Bytes:          Total: 25182080    Time: 60s   Cur: 25360022  422667/s
ep-B100:        Tx Bytes:          Total: 25182080    Time: 60s   Cur: 25359597  422659/s
                Rx Bytes:          Total: 0           Time: 60s   Cur: 0         0/s

lanforge@jedtest ~/scripts
 >
```

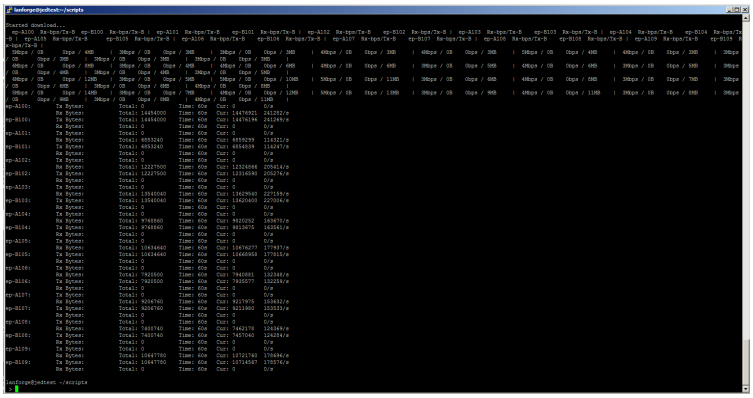
More Traffic Examples

1. Creating Multiple stations that transm

```
./lf_associate_ap.pl --resource 1 --mgr localhost \
   --action step1        --radio wiphy0    --ssid jedtest \
   --first_sta sta100 --num_stations 10 --duration 20 \
   --first_ip DHCP       --upstream eth1   --security wpa2 --passphrase jedtest1
```

2. Creating TCP/IP bursty traffic from 30Mbps to 450 Mbps

```
./lf_associate_ap.pl --resource 1 --mgr localhost \
   --action step1        --radio wiphy0    --ssid jedtest \
   --first_sta sta100    --num_stations 10 --duration 120 \
   --first_ip DHCP       --upstream eth1   --security wpa2 --passphrase jedtest1 \
   --cxtype tcp --bps-min 30Mpbs \
   --bps-max 450Mbps
```



3. Capturing that report with redirection

```
./lf_associate_ap.pl --resource 1 --mgr localhost \
   --action step1        --radio wiphy0    --ssid jedtest \
   --first_sta sta100    --num_stations 10 --duration 120 \
   --first_ip DHCP       --upstream eth1   --security wpa2 --passphrase jedtest1 \
   --cxtype tcp --bps-min 30Mpbs --bps-max 450Mbps &> report.txt
```

> i Both DOS and Linux command output can be saved to a file with the **&>** operator.

> i Both DOS and Linux files can be viewed with the **more** command.

4. Creating steady UDP traffic to at 450Mbps

```
$ ./lf_associate_ap.pl --resource 1 --mgr localhost \
   --action step1        --radio wiphy0    --ssid jedtest \
   --first_sta sta100    --num_stations 10 --duration 120 \
   --first_ip DHCP       --upstream eth1   --security wpa2 --passphrase jedtest1 \
   --cxtype udp --bps-min 450Mpbs \
   --bps-max SAME &> report.txt
$  more report.txt
```

5. Associating to an open AP

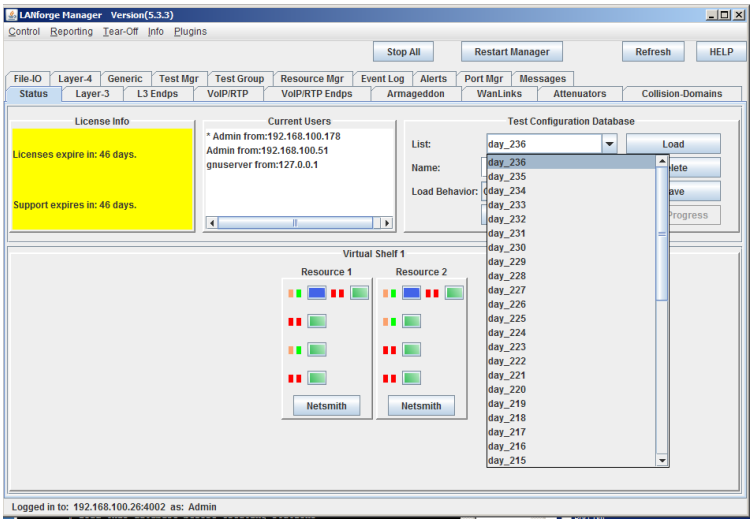
```
./lf_associate_ap.pl --resource 1 --mgr localhost \
   --action step1        --radio wiphy0    --ssid jedtest \
   --first_sta sta100    --num_stations 10 --duration 120 \
   --first_ip DHCP       --upstream eth1   --security open
```

6. Connecting a station at 802.11/abg speeds

```
./lf_associate_ap.pl --resource 1 --mgr localhost \
```

```
   --action step1        --radio wiphy0    --ssid jedtest \
   --first_sta sta100    --num_stations 10 --duration 120 \
   --first_ip DHCP       --upstream eth1 --security open \
   --wifi_mode abg
```

7. Initializing your test secenario by pre-loading a database. The database is the same name as the dropdown in the GUI Status tab.

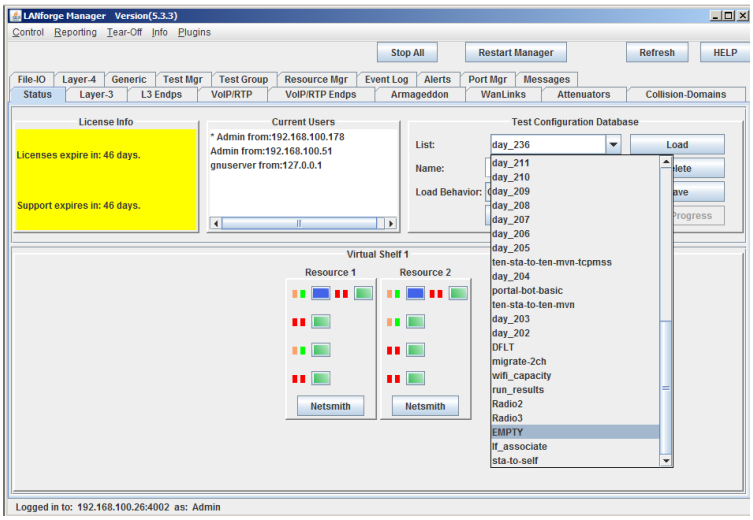



```
./lf_associate_ap.pl --resource 1 --mgr localhost \
   --action step1        --radio wiphy0    --ssid jedtest \
   --first_sta sta100    --num_stations 10 --duration 120 \
   --first_ip DHCP       --upstream eth1 --security open \
   --db_preload day_236
```

8. Saving your test state after completing a traffic run

```
./lf_associate_ap.pl --resource 1 --mgr localhost \
   --action step1        --radio wiphy0    --ssid jedtest \
   --first_sta sta100    --num_stations 10 --duration 120 \
   --first_ip DHCP       --upstream eth1 --security open \
   --db_preload day_236 --db_save station_results
```

9. Cleaning out your scenario settings after completing a traffic run. We can do this by loading the *EMPTY* database with the `db_postload` switch.



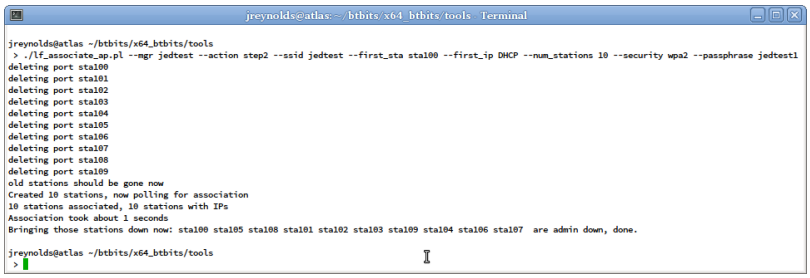

```
./lf_associate_ap.pl --resource 1 --mgr localhost \
   --action step1        --radio wiphy0    --ssid jedtest \
   --first_sta sta100    --num_stations 10 --duration 120 \
   --first_ip DHCP       --upstream eth1 --security open \
   --db_preload day_236 --db_save station_results --db_postload EMPTY
```

Using lf_associate_ap to stress test an AP

We can have a series of stations associate and unassociate over and over. This can be quite a bit of exercise for an AP. Below is a command that tests five clients connecting.

```
./lf_associate_ap.pl --mgr jedtest --action step2 \
 --ssid jedtest --first_sta sta100 --first_ip DHCP \
 --num_stations 10 --security wpa2 --passphrase jedtest1
```

This will create set of ten stations bring them up and then take them down.



Script Options

These might have been update since publication, please check --help output for your version of the script.

```
./lf_associate_ap.pl   [--mgr {host-name | IP}]
      [--mgr_port {ip port}]     # use if on non-default management port
      [--resource {resource}]    # use if multiple lanforge systems; defaults to 1
      [--quiet { yes | no }]     # debug output; -q

      ##       AP selection
      [--radio {name}]           # e.g. wiphy2
      [--ssid {ssid}]            # e.g. jedtest
      [--security {open|wep|wpa|wpa2}] # station authentication type
      [--passphrase {...}]       # implies wpa2 if --security not set
      [--wifi_mode {a|abg|abgn|abgnAC|an|anAC|b|bg|bgn|g}]

      ##       station configuration
      [--num_stations {10}]
      [--first_sta {sta100}]
      [--first_ip {DHCP |ip address}]
      [--netmask {255.255.0.0}]

      ##       connection configuration
      [--cxtype {tcp/tcp6/udp/udp6}]   # use a tcp/udp connection, default tcp
      [--upstream {name|eth1}]
         # could be AP or could be port on LANforge
         # connected to WAN side of AP
      [--bps-min {10000000}]         # minimum tx bps
      [--bps-max {SAME|bps-value}]  # maximum tx bps, use SAME or omit for SAME
      [--duration {30}]      # connection duration, seconds, default 60
      [--poll-time {5}]    # nap time between connection displays
      [--action {step1,step2}]
         # step1: creates [num_stations] stations and L3 connections
         # step2: does bringup test

      [--traffic_type {separate|concurrent}]
         # for step1: separate does download then upload
         # concurrent does upload and download at same time

      [--db_preload {scenario name}]
         # load this database before creating stations
         # option intended as a cleanup step

      [--db_save {name}]
         # save the state of this test scenario after running the
         # connections, before --db_postload

      [--db_postload {scenario name}]
         # load this database after running connections,
         # option intended as a cleanup step
```

Changing Station WiFi SSID with the CLI API

Goal: Programmatically change a stations SSID

Programatically creating LANforge virtual stations requires using the **add_sta** command. If you already have a station and need to change the SSID, you still use the **add_sta** command.

The general sequence of commands is:

1. if port is up, set port down with:

```
cur_flags=0x1 interest_flags=0x800002
```

2. issue add_port with changed SSID
3. issue set_port to bring it up with:

```
cur_flags=0x0 interest_flags=0x800002
```

We can create a station using this script command:

```
./lf_associate_ap.pl --action step2 --mgr jedtest \
  --resource 1 --radio wiphy0 \
  --ssid jedtest --first_sta sta100 \
  --num_stations 1 --first_ip=DHCP \
  --wifi_mode abgn --security wpa2 \
  --passphrase jedtest1 --quiet=0
```

The format of the add_sta command is listed in the CLI User's Guide. When we watch the debug output of the lf_associate_ap script, we see this **add_sta** command executed:

```
'add_sta' '1' '1' 'wiphy0' 'sta100' '1024' 'jedtest'
   'NA' 'jedtest1' 'AUTO' 'NA' '00:E3:F7:91:4A:1A' '5' 'NA'
   'NA' 'NA' 'NA' 'NA' '1024' 'NA' 'NA' 'NA' 'NA'
```

Looking at an example in the **lf_associate_ap.pl** script we see it being formatted here:

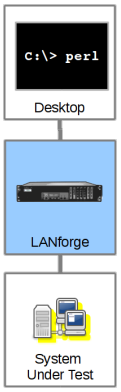
```
   my $sta1_cmd   = fmt_vsta_cmd($::resource, $::sta_wiphy, $sta_name,
                                 "$flags", "$::ssid", "$::passphrase",
                                 $mac_addr, "$flagsmask", $wifi_m);
   doCmd($sta1_cmd);
```

We format the parameters:

```
      return fmt_cmd("add_sta", 1, $resource, $sta_wiphy, $sta_name, "$flags",
                  "$ssid", "NA", "$key", $ap, $cfg_file, $mac,
                  $mode, $rate, $amsdu, $ampdu_factor, $ampdu_density,
                  $sta_br_id, "$flags_mask" );
```

Changing Station POST_IFUP field with the CLI API

Goal: Programmatically change a station's POST_IFUP field.
Creating a series of scripts using the lf_associate_ap.pl script is not adequate for negotiating a captive-portal environment, that script does not set the POST_IFUP parameter for the station. However, stations can be modified to gain that field.



Creating a station that negotiates a Captive Portal environment requires the POST_IFUP field to name a script. (Usually a *portal-bot.pl* script.) We can assign that port parameter with the **set_wifi_extra2** command. At the time of this writing, there are no perl scripts using this CLI command, but I will show an example here:

```
set_wifi_extra2,
   1,       # resource number
   sta100,  # port name
   0,       # flush-to-kernel
   NA,      # ignore probe
   NA,      # ignore auth
   NA,      # ignore assoc
   NA,      # ignore_reassoc
   NA,      # corrupt_gtk_rekey_mic
   NA,      # radius_ip
   NA,      # radius_port
   NA,      # freq_24
   NA,      # freq_5

   # post_ifup_script
   './portal-bot.pl --bot bp.pm --user "username" --pass "secret" --start_url "http:
   NA       # ocsp
```

The above command would never actually be formatted in in the way it appears above. It would all appear on one line without commentds.

In a perl script, the command could be formatted like:

```
my $cmd = fmt_cmd("set_wifi_extra2", 1,
         "sta100",  # port name
         0,         # flush-to-kernel
```

```
        "NA",       # ignore probe
        "NA",       # ignore auth
        "NA",       # ignore assoc
        "NA",       # ignore_reassoc
        "NA",       # corrupt_gtk_rekey_mic
        "NA",       # radius_ip
        "NA",       # radius_port
        "NA",       # freq_24
        "NA",       # freq_5
 qq(./portal-bot.pl --bot bp.pm ) # post_ifup_script
 .qq(--user "username" --pass "secret" )
 .qq(--start_url "http://www.google.com/" )
 .qq(--ap_url "http://localhost/" )
 .qq(--login_form "login.php" )
 .qq(--login_action "login.php" )
 .qq(--logout_form "logout.php"),
        "NA"        # ocsp );
```

Important Notes

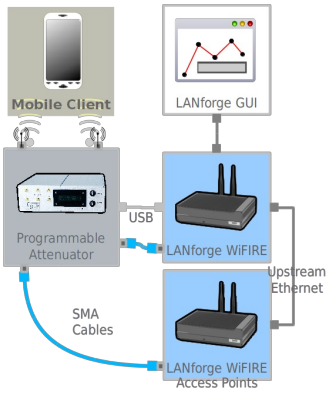
1. the LANforge server treats single-quotes (apostrophes, `'`) as command delimiters. Use only double-quotes (`"`) to quote the arguments to the script.
2. Do not use newlines (`\n` or carriage-returns (`\r\n`). That will truncate the command and LANforge will process it immedately.
3. These parameters will be provided by the server:

- `--mgt` The management FIFO
- `--ip4` The IP address of the port
- `--ip6` The IPv6 address of the port
- `--dnsv` A comma-separated list of DNS addresses
- `--logout` Signals logout

Generating a series of attenuations using data in a CSV file.

Goal: Using the attenuator_series.pl script and a specially formatted csv file, you will be able to re-play an arbitrary series of attenuations.

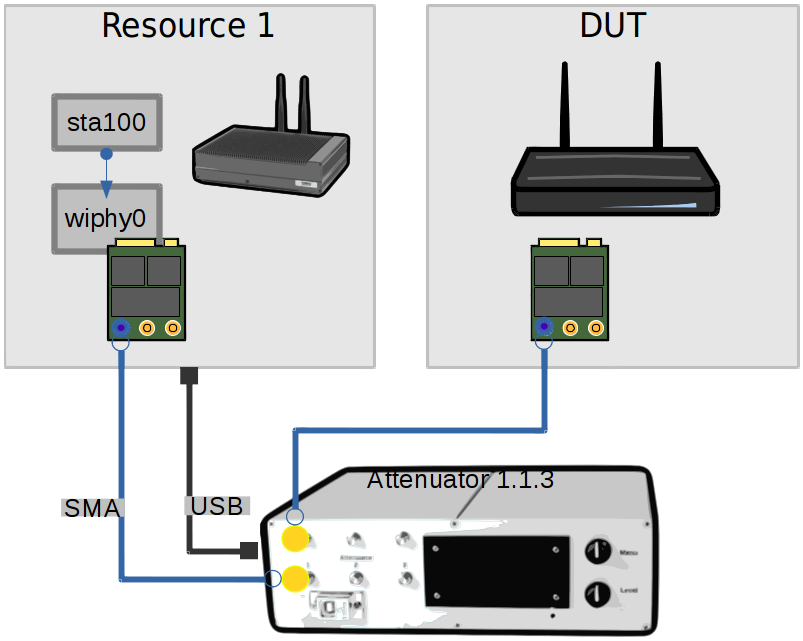
Playing back a series of WiFi attenuation levels using the `attenuator_series.pl` and a CSV file of attenuations make it possible to emulate the motion of a station (or stations) moving among a series APs. Or it could emulate interference in a crowd of moving people. Requires a LANforge CT703 (or better) and a LANforge CT520 (or better) system, and an access point.



Testing 1x1 with one attenuator

Our LANforge manager (resource `1.1`) has an attenuator serial number 3 (resource `1.1.3`) connected to the Device Under Test. The attenuator will be `1.1.1.3.` There will be station `sta100` on LANforge resource 1 and AP `vap0` on LANforge resource 2. Cables connect the radios to the the attenator. The radios are configured in 1x1 mode. The corresponding channel on the attenuator is `1.1.3.0`

[See LANforge Entity IDs for more on numbering.]



Let's script it with a simple data file: `/home/lanforge/atten_test1.csv`

```
channels,1.1.3.0
delay,5000
attenuate,250
attenuate,320
attenuate,450
attenuate,520
attenuate,820
```

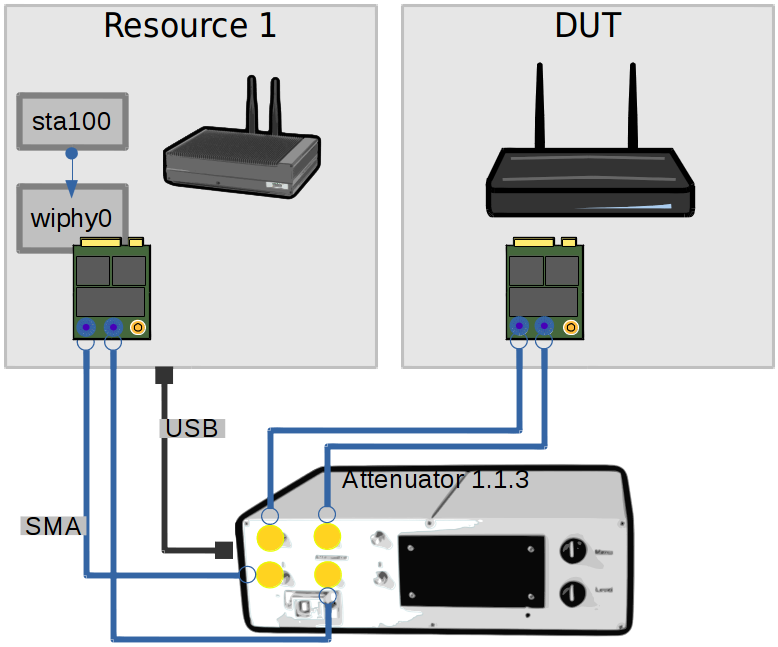
We run the script in our terminal:

```
$ cd /home/lanforge/scripts
$ ./attenuate_series.pl -f ../atten_test1.csv
```

Watching a Layer-3 connection in the Dynamic Display, we will see a dip, rise and dip at 10 second intervals.

Testing 2x2 with One Attenuator

Next we cable up the second channel (`1.1.3.1`). We can update the csv test file, by adding a new column for the channel.

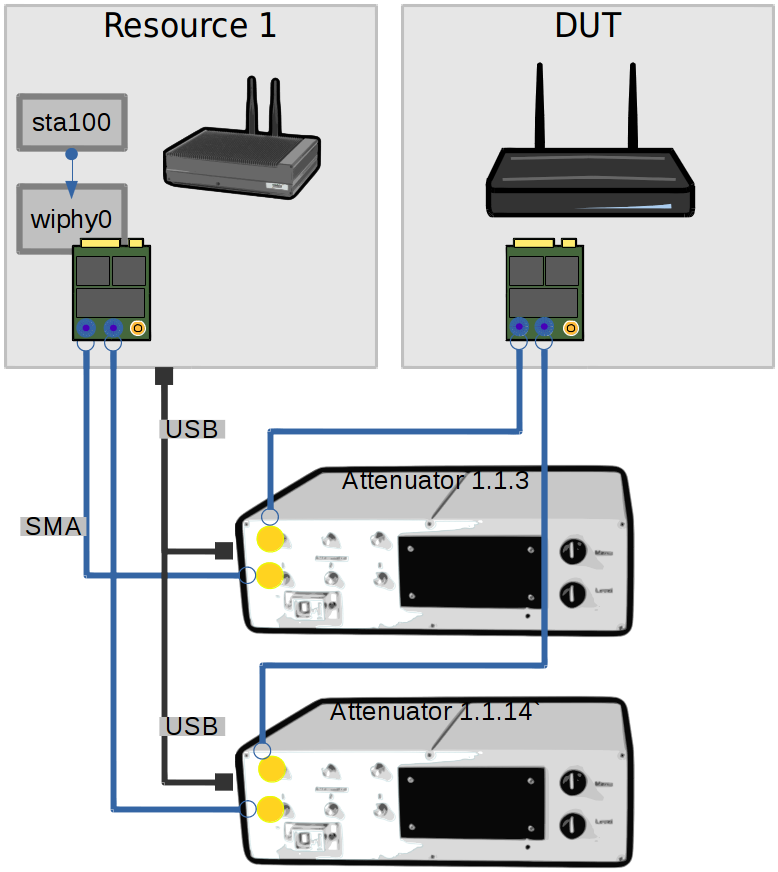



```
channels,1.1.3.0,1.1.3.1
delay,5000
attenuate,250,250
attenuate,320,250
attenuate,450,250
attenuate,520,520
attenuate,820,820
```

We can run the same command and watch the dynamic reports window to see a similar graph.

A 2x2 Example with Two Attenuators

The first radio on each LANforge is connected in 2x2 mode to both attenuators. This example is drawn to illustrate how you design the connection of your channels independently of their radios. Obviously, you don't need two attenuators for this scenario. However, if you had a CT523 with three radios and want to perform 2x2 testing with three client radios, it is possible to do so with only two CT703 attenuators.



We change the data file to specify the first channel on attenuator 14 (`1.1.14.0`):
`/home/lanforge/atten_test3.csv`

```
channels,1.1.3.0,1.1.14.0
delay,5000
attenuate,250,250
attenuate,320,320
attenuate,450,450
attenuate,520,520
attenuate,820,820
```

We can run the script once in our terminal:

```
$ cd /home/lanforge/scripts
$ ./attenuate_series.pl -f ../atten_test3.csv
```

Watching the port signal in the dynamic display we will see a rise and dip at 10 second intervals.

Connecting up Multiple Radios

There is no different in attenuator control whether you have one radio in 3x3 or three radios in 1x1 to control. If you are testing multiple radios, you will be monitoring their **RX Signal** in the dynamic report.

File Format

Editing the test data file with a basic spreadsheet program than can save to CSV format is possible. You will want to save with comma format, without double-quoting the cells. These directives are converted to lower-case, so you can type them in UPPER-CASE or Mixed-Case if necessary.

The format of the CSV file allows you to specify many options that might also be specified on the command line.

Directives

 # comments
: Rows that begin with a comment sign (`#`, `;`, `!`) will be entirely ignored. Cells in column B or beyond will be ignored.

channels
: Each cell following this directive specifies an attenuator channel to control.

sleep, nap
: The following cell specifies a one-time wait time in milliseconds

delay, naptime
: The following cell specifies a standard wait time in milliseconds between each `attenuate` command

attenuate, _, ,
: The following cells specify an attenuation value for channels specified by the last `channels` command.

minimum, min
: Sets the minimum attenuation permitted. Values below this will be set to the minimum directed.

maximum, max
: Sets the maximum attenuation permitted. Values above this will be set to the maximum directed.

Attenuation Values

- **Inherently Positive Values**, like `200` are absolute attenuation values, in deci-decibels. 200 means 20.0dB. The smallest unit of resolution is 0.5dB, so all your values will end in zero or five. E.G. (0, 5, 105, 200, 955). Values range between zero and 955.
- **Explicitly Positive Values**, that begin with `@+, ++, +` are increments with respect to the last value set on the channel.

  ```
  attenuate,250
  attenuate,@+50
  ```

  Results in the channel at 30.0dB. Spreadsheets often omit signed values when saving, so `@+` will force a text type cell.
- **Explicitly Negative Values**, that begin with `@-, --, -` are decrements with respect to the last value set on the channel.

  ```
  attenuate,300
  attenuate,@-50
  ```

  Results in the channel at 25.0dB. Spreadsheets often omit signed values when saving, so `@-` will force a text type cell.
- **Basic Cell Math** can be performed, but only against absolute cell values.

  ```
  attenuate,500,400
  attenuate,=B1+50,=C1-50,   # results in 550, 350
  attenuate,=B2+5,=C2-5,     # fails: B2 and C2 were formulas.
  ```

  This feature is unlikely to be as useful as it sounds, because pasting a column of forumae will be pretty useless, since a spreadsheet processes them recursively. Also, most spreadsheets saved to CSV typically don't save formulae by default, you probably will get the computed values in your CSV file.
- Shortcuts include `_, NA, and ,,`. You can skip a computation on a cell by leaving a blank cell, underscore, or 'NA'. Careful: the value +0 will likely be truncated to 0, and set the channel to 0.0dB attenuation.

Script Options

The `attenuate_series.pl` script uses these arguments. They support long and short argument switch names:

```
  -m
  --mgr             LANforge manager host, like localhost or 192.168.101.1
```

```
-f
--file            CSV file with attenuation data

-d
--delay           Override of DELAY variable, milliseconds between applying rows

-l
--loop            Repeat indefinitely

-c
--channel         Override of channels variable, eg: 1.2.3.1,2.3.4.3

-i
--min
--minimum         Set minimum attenuation value (not lower than zero)

-x
--max
--maximum         Set maximum attenuation value (not higher than 955)

-n
--dry
--dryrun
--dry_run         Do not apply attenuation, just parse file, ignore nap times
```

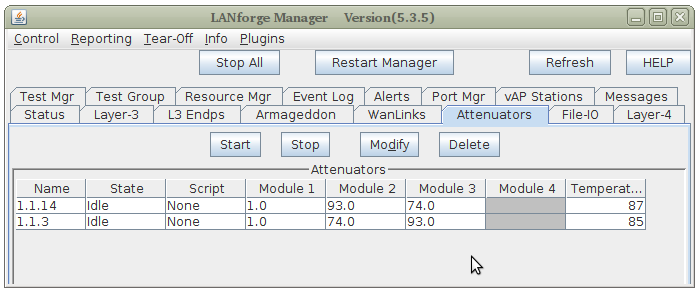
Example CSV File

This CSV shows a working example that gives warnings.

```
# example csv
channels, 1.1.14.0, 1.1.14.1, 1.1.14.2, 1.1.3.0, 1.1.3.1, 1.1.3.2,
DELAY,2000,,,,,,
ATTENUATE,950,850,750,950,850,750,
attenuate,940,-10,-10,-10,-10,-10,
attenuate,930,-10,NA,-10,-10,,
attenuate,=B4-10,=C4+10,NA,-15,-15,,
attenuate,-15,_,-15,,NA,-15,
sleep,1000,,,,,,
attenuate,110,115,215,315,415,515,
_,=B10-20,=C10+20,=D10+20,=E10+20,=F10+20,=G10-20,
_,@+10,@+10,@-10,10,10,10,
# eof
```

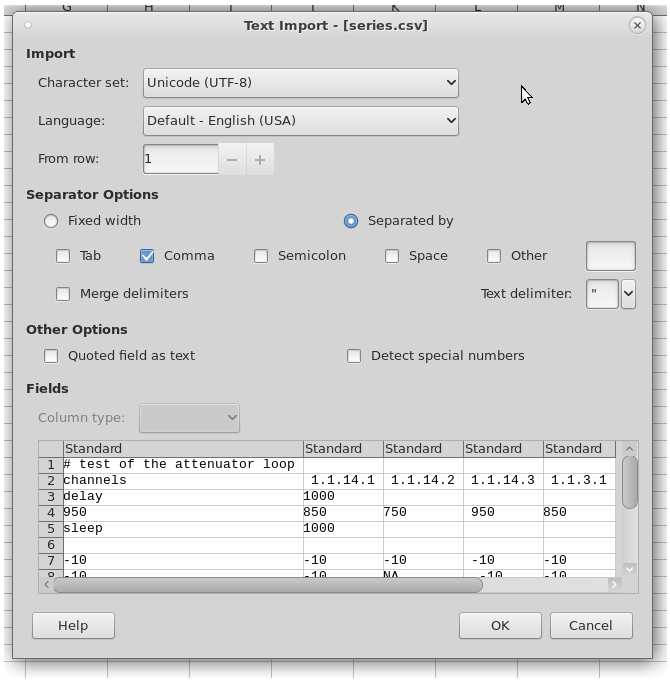
Attenuators Tab

Here's the Attenuators tab used for the examples:

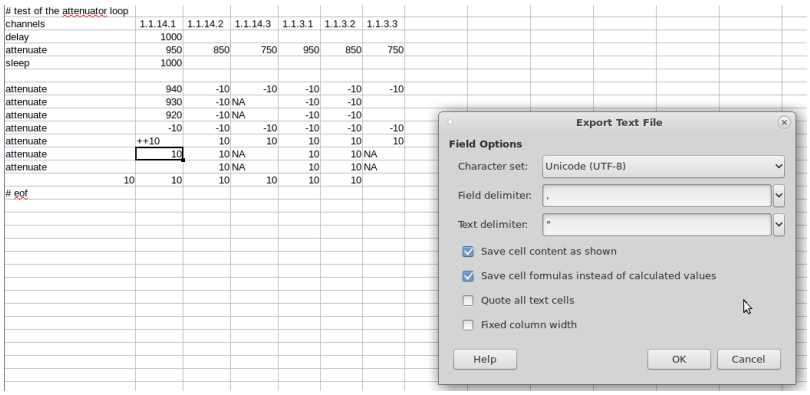


Opening and Saving CSV

Here are options used for the open dialog in LibreOffice Calc:



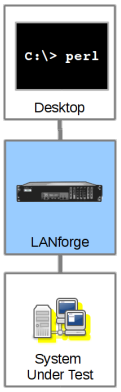
Here are the options used for the save dialog in LibreOffice Calc:



LANforge Entity IDs

Goal: Gain a better understanding of LANforge Entity IDs (EIDs)

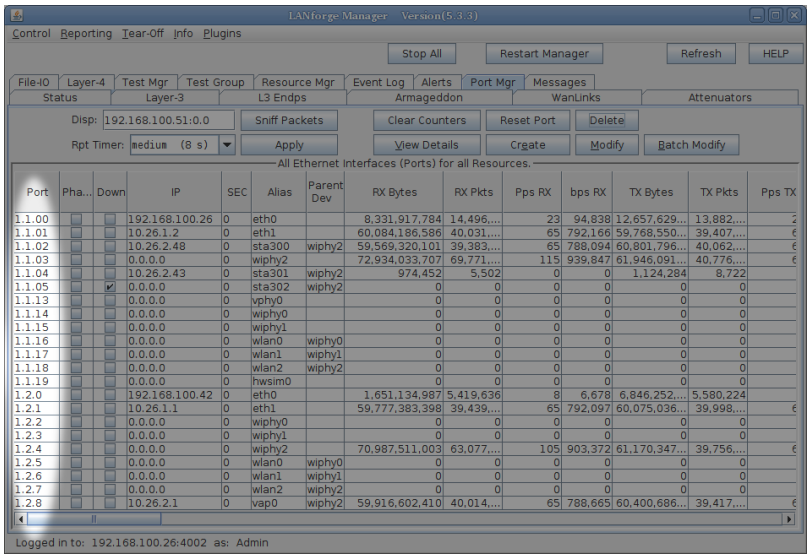
Every port, radio, virtual port, endpoint and connection in LANforge has an ID known as an EID. These are an internal notation that expresses the hierarchy of the physical and virtual objects managed by LANforge realm.



Ports, Endpoints and Connections are Entities

Entity IDs (EIDs) are a dotted-decimal phrase. It expresses the Shelf, Resource Number, Port or Connection number, and if it is an endpoint, it gains a fourth decimal. An example:

```
1.2.8.4   : EID
1         : shelf
  2       : resource
    8     : port
      4   : endpoint
```



For now, assume the shelf number will always be 1. The Resource number will refer to the LANforge machine ID as reported on the Status tab. The Port ID is only unique within a LANforge machine. The Port ID also refers to hardware in a machine: radios get a third decimal. The fourth decimal refers to either endpoints or connections.

Only Some LANforge Entities Generate Connection Data

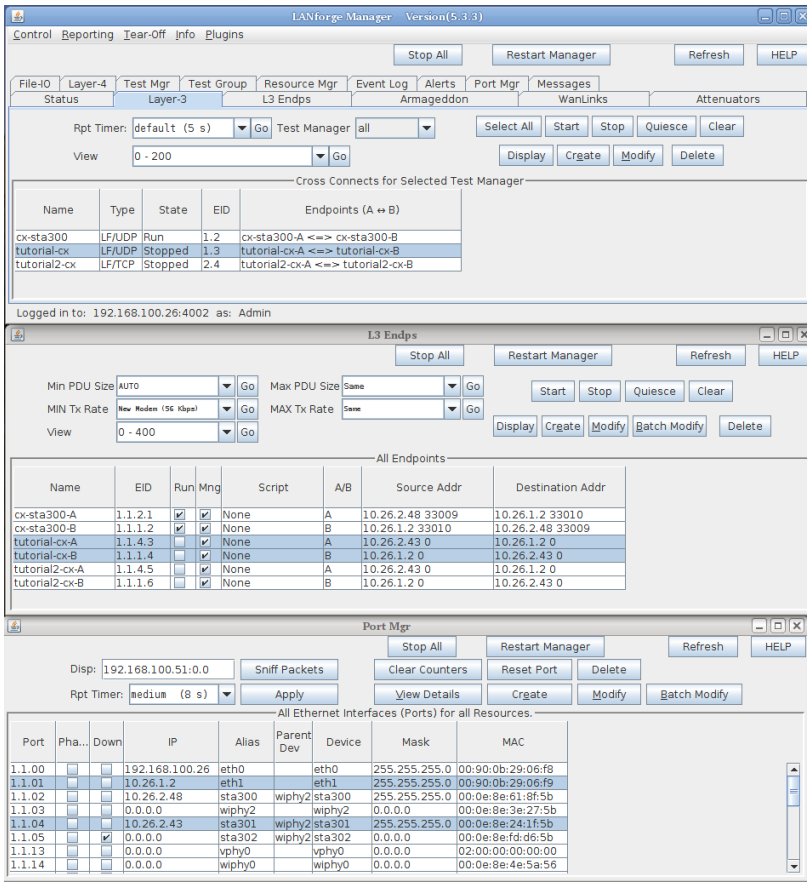
While some items with port numbers, notably radios and ports, do not generate traffic. Endpoints generate traffic, and typically endpoints are transmitting to an opposite endpoint. The exception to this are *multicast endpoints*.

EIDs Express Hierarchy

From the dotted-decimal perspective:

- Physical or virtual ports reside below a resource, except:
- ...for VLANs: A virtual port does not reside below it's physical port
- ...for bridge ports:: A port of a bridge has to exist before the bridge is created
- An endpoint resides below a physical or virtual port.

The formatting of the decimals might or might not be zero-padded. The picture below should convey how a connection (Layer 3) relates to two endpoints, and two ports:



The exception is connections. Connections are numbered outside of this hierarchy.

Do I use EIDs in Scripts?

Usually not, for these reasons:

1. EIDs are generated at LANforge manager start time, and might depend on the detection order of

ports when the PCI bus on the host is enumerated at boot time.

2. New EIDs can be created by appending one database to another on non-conflicting devices
3. New devices can be hot-added to a LANforge resource, like a programmable attenuator or a USB-Ethernet adapter, generating new Port IDs.

In scripts, it *is legal* to reference port numbers, but not advised to store them between sessions. If you reference an EID, it should be from within your present LANforge session. If your resources tend to disappear off the network and return (you had a machine reboot) those EIDs are not guaranteed to return.

> i For ports, only the first two decimals (shelf and resource id) are actually stable across machine reboots.

If you look into the saved scenarios (in `/home/lanforge/DB/DFLT`) you will notice that ports, endpoints, and connections are refered to by name. Event though in the CLI Users's guide, where it states *port number*, use names in your scripts:

```
CMD
|         SHELF
|         |  RESOURCE
|         |  |  PORT
|         |  |  |
set_port 1  1  eth1 10.26.1.2 255.255.255.0 10.26.1.1 ....
```

First-time User Introduction to LANforge: Scripting and GUI

Goal: This outline is a rough and generic overview of our GUI. This outline, that references other Candela Technologies documentation on our website, briefly covers basic GUI tasks and traffic generation that may be shown to a new customer whom has never used the GUI before, without overloading them with great detail.

---

1. **Table of Contents**

   A. Basic GUI port manager layout and introduction
      A. Editing the GUI tabs and Port Manager to display relevant information
      B. Changing Columns in the Port Manager
   B. LANforge GUI Tab Introduction
      A. Status
      B. Port Mgr
      C. Layer-3, Layer-3 Endps
      D. Layer 4-7
      E. Resource Mgr
      F. Messages, Warnings, Wifi Messages
      G. Using Netsmith
   C. Station Creation
      A. Searching for Active SSIDs & Connecting to a Particular SSID
   D. MAC-VLAN Creation
   E. Bridge Creation
   F. Virtual AP (VAP) Creation
   G. Monitor Creation
   H. Layer-3 Cross-Connection
   I. Layer-4 Cross Connection
   J. Introduction to Chamber View & Running Scripts in Chamber View

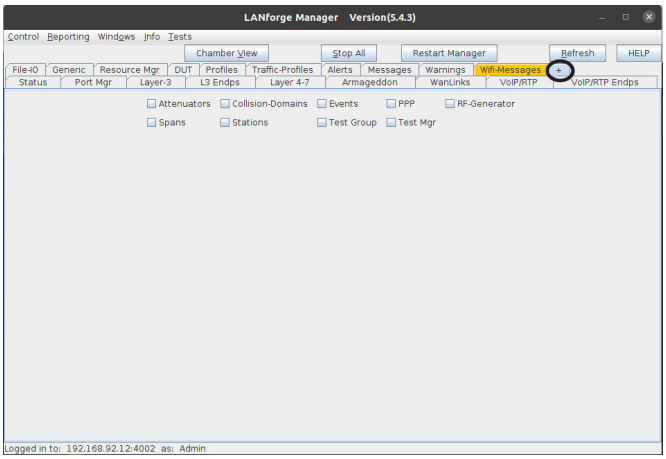
2. **Basic GUI Port Manager layout and introduction:**
   - After connecting the GUI, the interface will automatically open to the *Status* page. There are 28 tabs/pages that the GUI has, not including the *Netsmith* View and the *Chamber View*.

A. **Editing the GUI tabs and *Port Manager* to display relevant information**

- Upon opening the GUI, several default GUI tabs open as well. Depending on what upcoming WiFi testing must occur, more (or less) GUI tabs may need to be open than the ones defaulted.
- When running python scripts aimed to automate the GUI, the tabs that the actions in the script are occuring in must be displayed in the GUI (unless the user is running the GUI in headless mode).

A. To **display** tabs that are hidden:

- Click on the *+* tab under *Refresh* in the top right hand corner. Then, select which tabs to add to the GUI display.

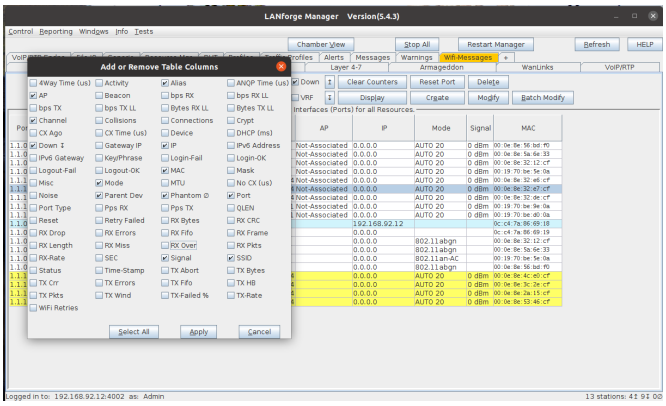


B. To **hide** tabs that are currently displayed:

- Right-click the mouse on any tab that is aimed to remove and click *Hide*. This is will remove the tab from the GUI interface currently and will be placed under the *+* category.

B. **Customization of Column Display in the *Port Manager***

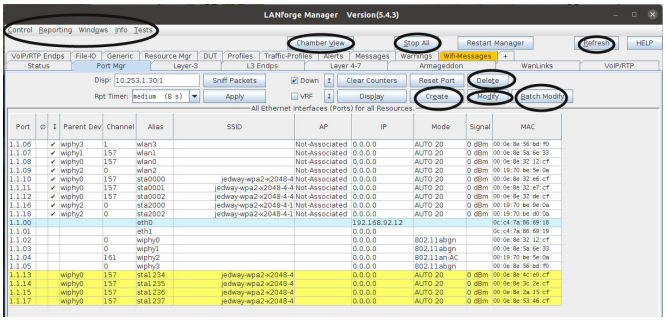
A. In the second tab, *Port Manager*, comes downloaded with all the tab columns selected to be displayed (73 columns). To change which columns are selected and displayed, Right-click the mouse in any column space and select *Add/Remove Table Columns*. From that point, select the necessary columns wished to be displayed in the *Port Manager*.



B. After selecting the columns that wish to be displayed, Right-click the mouse again in the body/rows of the *Port Manager* and select *Save-table Layout*. This will make sure the changes don't revert the next time the GUI is opened and closed.

C. After resizing, one can also Right-click the mouse in the body/rows of the *Port Manager* and select *Auto-size*, to auto-size the columns to make sure that all the words under each column are in vision at first glance.

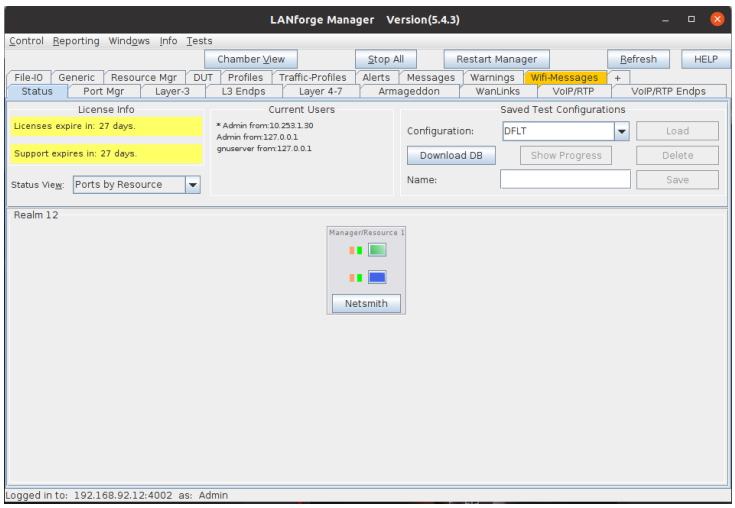
D. Tip: hot-keys are enabled throughout the entirety of the GUI. In some places in the GUI, there are lines underneath some letters in buttons. To use the keyboard shortcut for that button, press Alt + that letter underlined in the word to press the button. This also works for drop-down menus when the shortcut is enabled via an underlined letter in a word. **Note:** MAC users need to use key combo ctl+alt + letter to do shortcuts. Circled below are some examples of hotkeys enabled.



3. **LANforge GUI Tab Introduction**

A. **Status** tab:

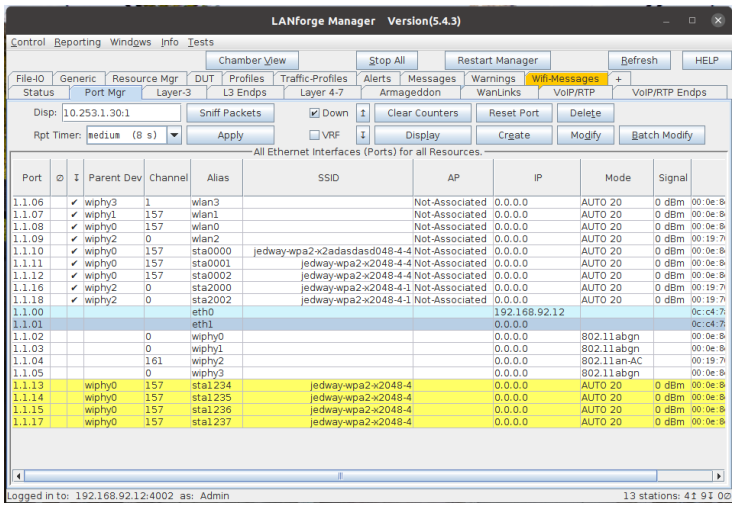
Please read the see also below (LANforge Manager) to read about the LANforge Status tab. This is where information about the server is typically stored, configurations of the GUI are able to be saved, and where the *Netsmith* is.



For more information see Step 2: LANforge Manager

B. **Port Mgr** tab:

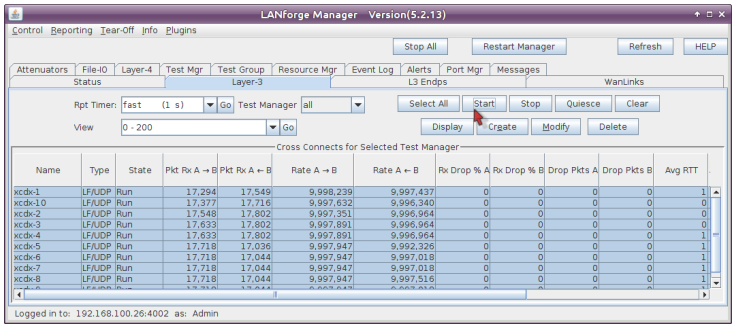
The *Port Mgr* tab is where all the ports and representations of the radios, wifi objects, and ethernet connections are located. The *Port Mgr* (or *Port Manager*) includes the location/appearance of all further MAC-VLANs, 802.1Q-VLANs, Redirects, Bridges, Bonds, GRE Tunnels, WiFi Stations, WiFi VAPs, WiFi Monitors and WiFi Virtual Radios. Please read more about the *Port Mgr* tab next to *see-also* below



For more information see Ports (Interfaces)

C. **Layer-3** tab, **L3 Endps** tab:

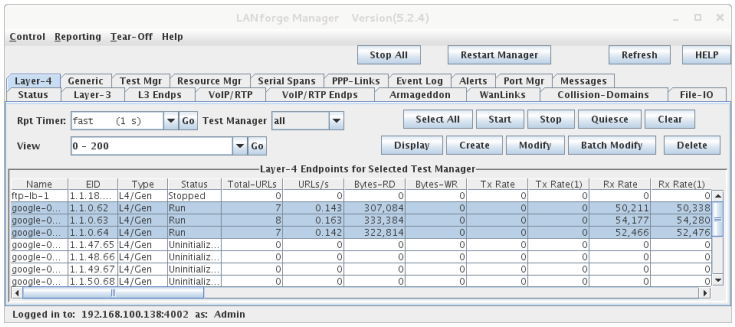
The *Layer-3* tab are where Layer-3 WiFIRE traffic connections are made, started, stopped, modified, and displayed. Each cross-connects have 2 endpoints each. These endpoints and the traffic/data associated with them are found and elaborated under the *L3 Endps* tab in the GUI. Please visit the introduction to Layer-3 Cross-Connects, linked below, for a general overview.



For more information see Layer-3 Cross-Connects (FIRE)

D. **Layer 4-7** tab:

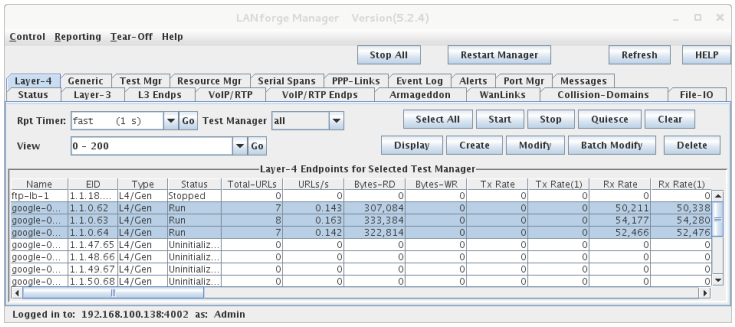
The 'Layer 4-7' tab is currently where *Layer-4* HTTP, HTTPS, FTP, FTPS, TFTP, SCP and SFTP endpoints are made. These are stateful protocols that will communicate properly with third-party servers. FTP, FTPS, TFTP, SCP and SFTP can upload and download, and the other protocols are only for downloading. The Layer 4-7 tab is used to manage Layer 4-7 endpoints.



For more information see Layer 4-7

E. **Resource Mgr** tab:

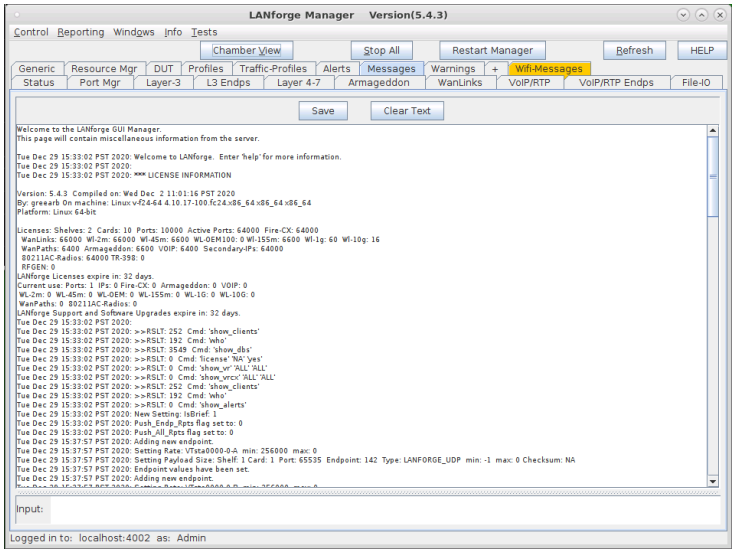
The *Resource Mgr* tab displays information on all Resources discovered by the LANforge server and provides the ability to perform system functions on selected machines (one or more). The definition of a resource is a LANforge machine that belongs to a numbered realm. The realm 255 is always a stand-alone realm while the realm resource 1 is the manager. The Resource Mgr tab displays LANforge servers in the same realm. LANforge systems have to be manually numbered, two LANforge systems with the same resource ID will confuse the manager resource. Please visit the link below for more information on the *Resource Mgr*



For more information see Resources (Data Generator Machines)

F. **Messages, Warnings, Wifi-Messages Mgr** tab:

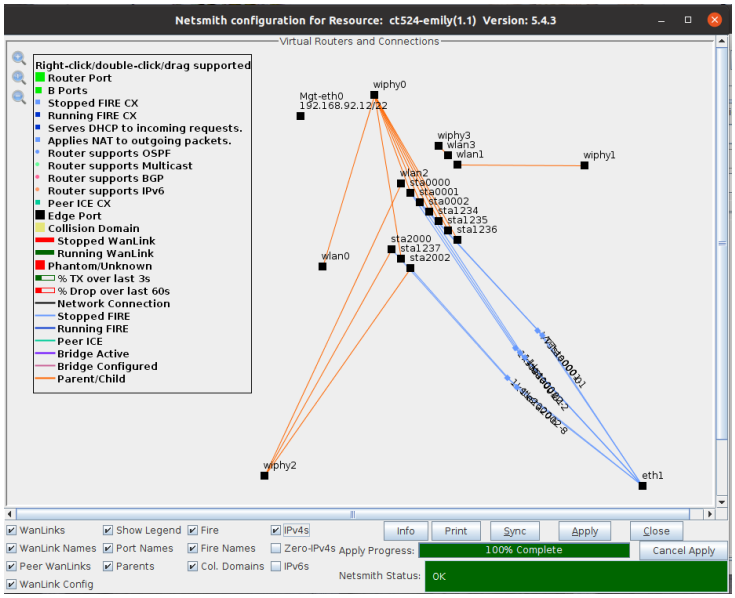
The *Messages*, *Warnings* and *Wifi-Messages* tab are all tabs that should be open at all times. All these tabs contain important information about the LANforge GUI Interface. The *Messages* tab displays detailed CLI command feedback from the LANforge Server. When scripting, command failures can be shown here. If <u>any</u> one of these 3 tabs are highlighted/have a yellow background in the tab bar, there is a new update in that yellowed tab.For information on any other tabs, besides the ones mentioned above, please visit the link below *LANforge-GUI User Guide: Tab Display Preferences* for further tab descriptions.



For more information see Tab Display Preferences

G. **Using Netsmith** tab:

In the LANforge GUI, on the *Status* page there is a small button named *Netsmith*. It is a tool used to help visualize the relationships of ports and cross connects defined in the resource you are viewing. There is a separate *Netsmith* view for each LANforge resource in your realm. There are several ways to edit the GUI objects in *Netsmith*, display the different up-to-date connections in the GUI, and what is shown in *Netsmith*. Please visit the link below to understand how to use *Netsmith* in greater detail.



For more information see Netsmith: Virtual Network Configurator

4. **<u>Station Creation:</u>**

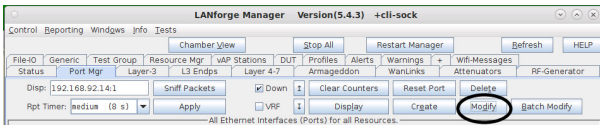
Please visit Step 1 of the following cookbook below to learn how to Create a Station in the LANforge-GUI. Please visit the link at the bottom of this section on how to script a station in the GUI.

A. **Searching for Active SSIDS & Connecting to a Particular SSID**:

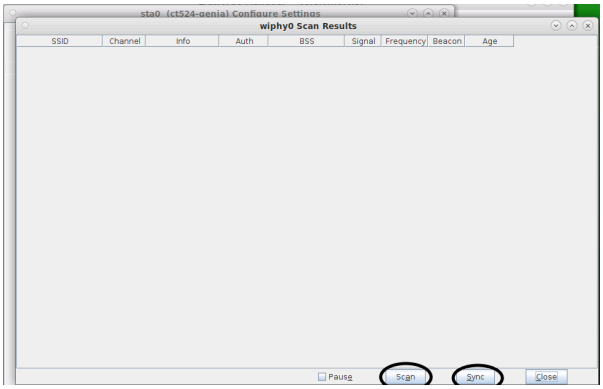
Often times, there may be an active network around, but the LANforge GUI does not have the network registered as "able to be connected to". To allow this network to be recognized, one must **scan** in the GUI object's settings to make sure that the object sees this network.

See below for an example:

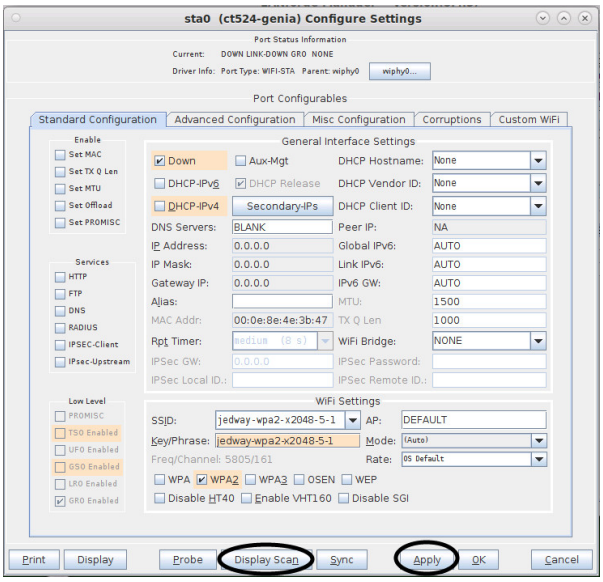
I. Double-click or select *Modify* on a station in the Port Mgr to pop up *Configure Settings* window.



II. However, the GUI isn't registering that as a proper network because selecting *Display Scan*, *Scan*, and *Sync* in the *Configure Settings* shows no networks are found and discovered in the GUI.

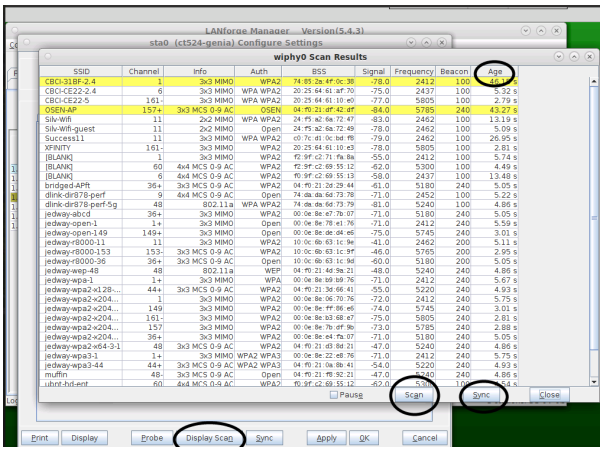


III. Type in the desired SSID, Key/Phrase, and select the appropriate Security to be used (WPA/WPA2/WPA3... etc) located within the *WiFi Settings* panel (shown below). Select *Apply*. *Apply* will trigger the LANforge GUI to start searching for currentlyactive SSIDs.

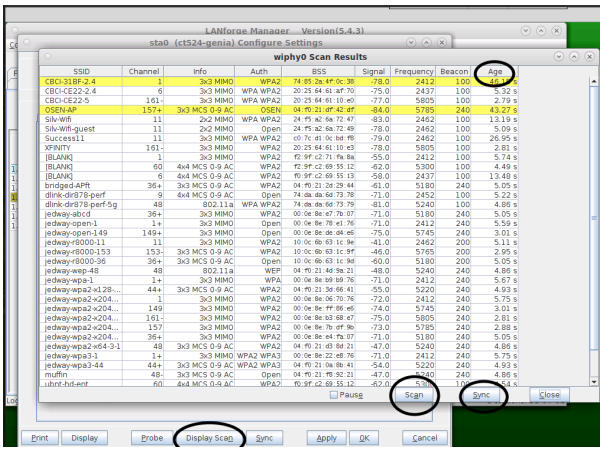


IV. Then, select *Display Scan* in the bottom bar, as highlighted in the picture above. Something similar to the Window in the picture below will pop up. Then click on *Scan* (circled below) and *Sync*. Now, the most recent active networks should be scanned and displayed in a similar window to below by the GUI. The example below indicates that the radio *(wiphy0)* has now found current, active networks. Also, the far right corner of the table displays the age of the networks, so if the *Age* is too old after the recent scanning, it might be time to restart the network or pick a new network.

Note: If there are no scan results, the radio is probably set to a specific channel. The radio channel configuration may need to be changed or the object must be created on a different radio.



V. . Now, close the two windows opened previously by selecting *Close*. Go back to the *Port Mgr* tab and the desired object to be connected should be connected to that SSID. In *Wifi-Messages*,there should have also been a message saying that sta0 and wiphy0 are scanning for network SSIDs. This is another indication of the LANforge scanning software retrieving local SSIDs. LANforge now concludes that it can connect to the SSID by acquiring an *AP* and *IP* in the *Port Mgr* (see circled below).



For more information see Station Creation : Step 1

For more information see Scripting a Station in the GUI (Step 3)

5. **MAC-VLAN Creation:**

Creating a MAC-VLAN on the LANforge-GUI is done in the *Port Mgr.*

Please visit **Step 3** of the following cookbook on how to create a MAC-VLAN from the GUI. The following link will inform how to program the GUI to create a MAC-VLAN

For more information see Creating a MAC-VLAN in the GUI(Step 3)

For more information see Scripting a MAC-VLAN in the GUI

6. **Bridge Creation:**

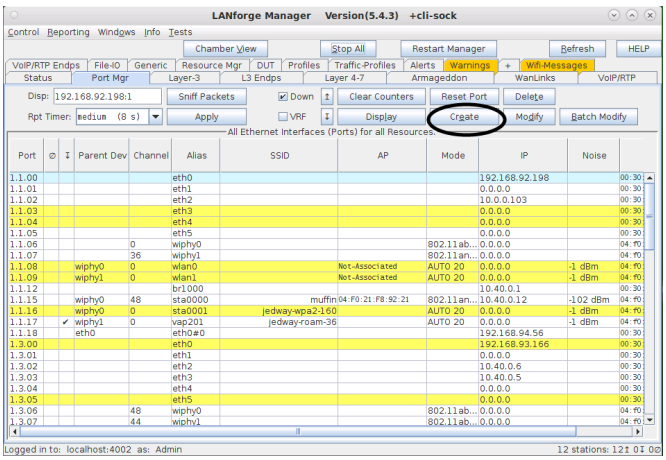
Creating a Bridge on the LANforge-GUI is done in the *Port Mgr.*

Please visit **Step 2** of the following cookbook on how to create a Bridge in *Netsmith*.

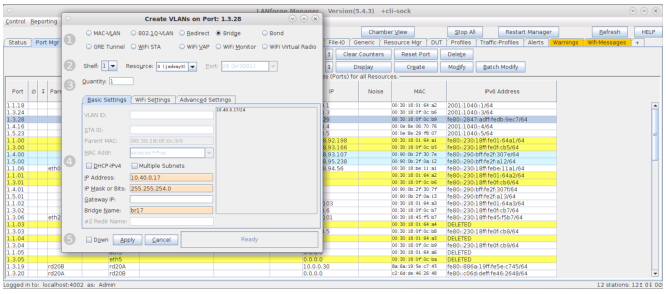
For more information see Creating a Bridge in *Netsmith* (Step 2)

A. Create a bridge in *Port Mgr.*

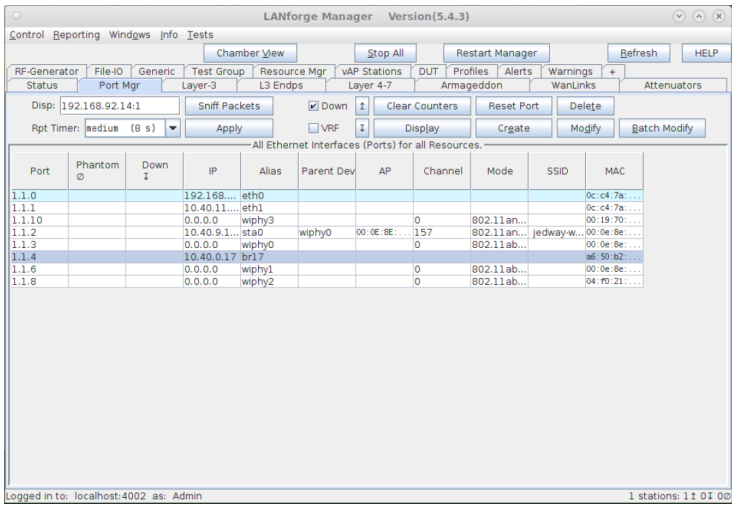
A. Click on the *Port Mgr* tab and Create in the top right corner.



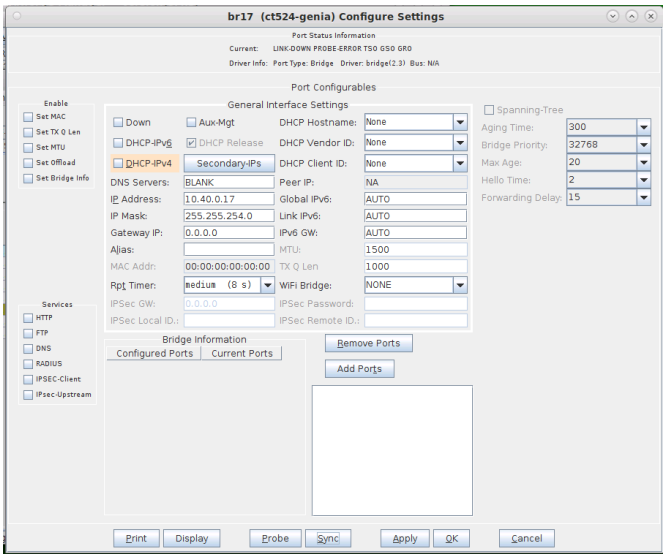
B. After a new window pops up, Select *Bridge* in Step 1 of the new window. In Step 2, select the *Shelf* and *Resource* the bridge should use (from the drop down menus in each slot). Step 3, select the *Quantity* of the bridges to be created. In Step 4, under the *Basic Settings* tab, check the box if the bridge should be enabling *DHCP-IPv4*. If *DHCP-IPv4* isn't enabled, give the bridge an *IP* Address and *IP Mask*. Lastly, give the bridge a name, Click *Apply* and *Cancel*. The bridge is now in the *Port Mgr.*



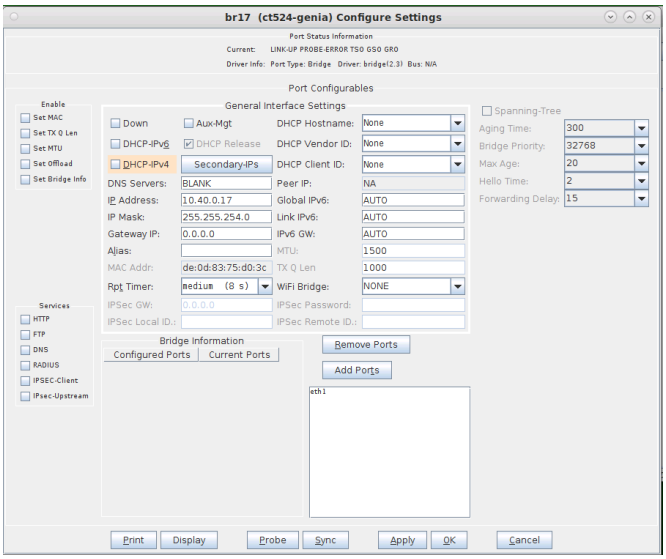
B. Adding a port to an existing bridge in *Port Mgr*:



A. To add a port, double click on the bridge you created or click once on the bridge in *Port Mgr* and select *Modify*. A window *Configure Settings* should pop up. At the bottom of the window, there is a small section that allows addition of ports.



B. In the text box under the *Add Ports* button (circled below), type in the port name (ex: vap123, eth1, sta000) intended to be added to the bridge. In this example, *eth1* to be added to *br17*.



C. Select *Add Ports* (circled). This button will now categorize *eth1* as a *Configured Port*. Then, select *Apply* and *Sync* to now see *eth1* also be listed under *Current Ports*. Lastly, click *OK* to close the window. If the port inputted into the text box does not move to the *Current Ports* category after selecting *Sync*, this may mean that the port is already in a configuration that prevents it from being in a bridge (i.e. it may already be in a bridge…etc). To learn how to script a bridge in the GUI, please visit the link below.

For more information see Scripting the GUI to create a Bridge (Step 6)

7. **Virtual Creation (VAP):**

   Please visit **Step 1** of the following cookbook to learn how to create a Virtual AP in the GUI. For more information see Scripting the GUI to create a VAP (Step 7)

8. **Monitor Creation:**

   Please visit **Step 1** of the following cookbook to learn how to create a Monitor in the GUI. For more information see Scripting the GUI to create a Monitor

9. **Layer 3 Creation**:

   Layer-3 Cross-Connects represent a stream of data flowing through the system under test. A Cross-Connect (CX) is composed of two Endpoints, each of which is associated with a particular Port (physical or virtual interface). The *Layer-3* tab displays connections 0-200 by default.

   Separated below are important sections to getting to know the *Layer 3* tab:

   For more information see How to Create and Modify Cross-Connects & Cross-Connect Information

   For more information see Interpreting the Layer-3 Endps tab: Layer-3 Cross Connect Endpoints & Batch-Creating Cross-Connects

   For more information see Scripting a Layer-3 Cross Connect (Step 8)

10. **Layer 4-7 Traffic Generation**:

    The Layer 4-7 traffic is supposed to emulate curl commands. Endpoints can be created with the following protocols: HTTP, HTTPS, FTP, FTPS, TFTP, SCP and SFTP. These are stateful protocols that will communicate properly with third-party servers. FTP, FTPS, TFTP, SCP and SFTP can upload and download, and the other protocols are only for downloading. The Layer 4-7 tab is used to manage Layer 4-7 endpoints.

    Separated below are important sections to getting to know the Layer 4-7 tab:

    For more information see Creating and Modifying Layer 4-7 Endpoints, L4 Endpoint Information, Batch-Create Layer 4-7 Endpoints

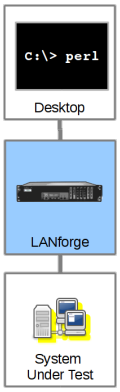
    For more information see Layer 4-7 Endpoint Display

    For more information see Setting up a Simple HTTP Get/Download in the GUI

    For more information see Scripting the GUI to create Layer 4-7 traffic (Step 9)

Querying the LANforge GUI for JSON Data

Goal: The LANforge GUI now has an embedded webserver and a headless mode of operation. Read on for how to configure and query the client for JSON formatted data.



Updated 2019-11-21: *New features in 5.4.1.*

- Some of the CLI API parameter names have changed. Notably: **nc_show_ports** `flags` changed to `probe_flags`. Be aware that older scripts might break on upgrade.

Updated 2018-07-24: *New features in 5.3.8.*

The LANforge GUI (as of release 5.3.6) can be configured to start an embedded web server that can provide data about ports and layer-3 connections. This service can be queried with with any browser or AJAX connection. We're going to increasingly refer to it as the LANforge **client**. This feature provides these benefits:

- More rapid polling: using CLI scripts to poll ports on the LANforge manager can add stress and contention to the LANforge manager; polling the GUI will not tax your test scenario.
- Expanded array of data: the views found in the GUI, like Port Mgr and Layer-3 tabs, contain synthesized data columns not available through the CLI scripting API. Most of these columns can be returned in JSON format.
- Reduced effort when integrating with third party test libraries: many other testing libraries expect JSON formatted input.
- Web socket delivery of event data allows real-time reporting of interface changes and station scan results. This is also a channel for querying additional diagnostic data.
- There is a /help web page that allows you to build POST commands.
- A headless `-daemon` mode that will run the client without any GUI windows. This requires much less memory and has been queried for weeks at a time without crashing or memory leaks.

Present and potential drawbacks of the JSON feature:

- Actively being developed: the JSON views/schema of the objects is at a demonstation state. URLs and JSON structures have changed in 5.3.8.
- Now no longer possible to create Groovy plugins to add JSON features if you want to use the headless mode. JSON Features are compiled into the LANforge GUI from Java sources.
- In 5.3.8 we have limited the view of ports, have added URLs to post direct CLI commands, and have applied HTML `application/x-www-form-urlencoded` form posting submissions in name/value pairs. There is no `multipart/form-data` JSON submission at this time.

Client Settings

The LANforge GUI is started using a script (`lfclient.bash` or `lfclient.bat`). From a terminal, we call that script with the `-httpd` switch. By default the GUI will listen on port 8080:

```
$ cd /home/lanforge
$ ./lfclient.bash -httpd
```

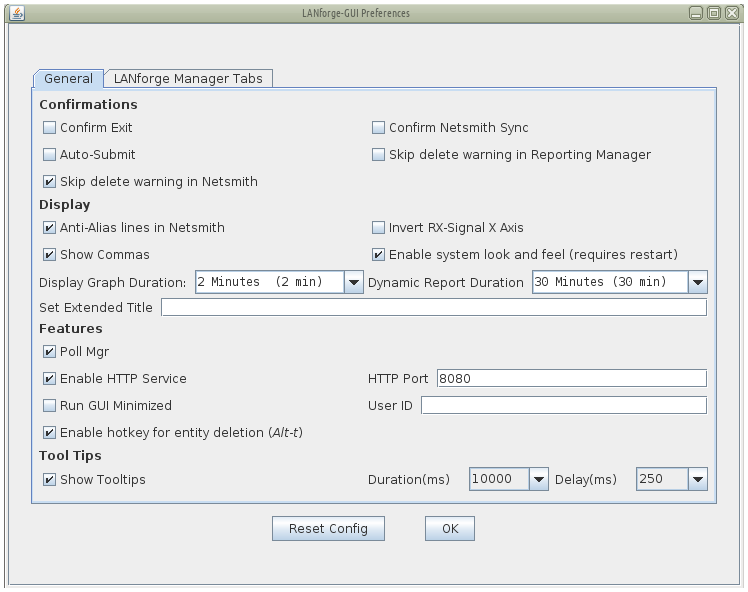
You can specify the port to listen on:

```
$ ./lfclient.bash -httpd 3210
```

You can run the client headless with the `-daemon` switch as well:

```
$ ./lfclient.bash -httpd -daemon
```

There is a setting in the 5.3.8 *Control→Preferences* menu for setting a *minimized mode* and the HTTP port number as well.

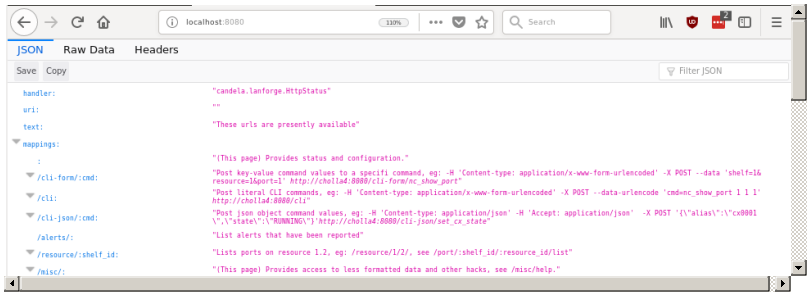


Making Queries

From the terminal we can query the port to find a basic message from the GUI:

```
$ curl -sq http://localhost:8080/
```

This first page (/) will give you a JSON list of the resource URLs available. Most URLs will provide JSON as their default content type. Notably, `/help` defaults to HTML.



By default, most URLs will treat a default `Accept: */*` header as `text/html`. Compare the two techniques below:

JSON Output

```
$ curl -sqv -H 'Accept: application/json' http://localhost:8080/resource/1/1
{"handler":"candela.lanforge.HttpResource$JsonResponse","uri":"resource","candela.lanforge.Htt
```

Clearly, the JSON output is difficult to read. We cover formatting output below.

HTML Output

Most of the queries to the client will return JSON by default. The notable exception is the `/help` URL. To get HTML output in the terminal, you have to specify `Accept: text/html` to curl:

```
$ curl -sqv -H 'Accept: text/html' http://localhost:8080/port/1/1/1
<!DOCTYPE html>
<html>
<head><title>/port</title>
</head>
<body>
<table border='1'><thead><tr><th>EID</th><th>AP</th><th>Activity</th><th>Channel</th><th>Devic
<tbody>
<tr><td>1.1.1</td><td></td><td>0.0</td><td></td><td>eth1</td><td>false</td><td>0.0.0.0</td><t
</table><hr />
</body>
</html>
```

Formatting Results

JSON formatted text is pretty difficult to read, there are a few different utilities that can help you look at it: `jq, json_pp, json_reformat, tidy, xmllint, yajl` and `jsonlint`.

Example of installing formatters

On Fedora, install:

```
$ sudo dnf install -y jq perl-JSON-PP tidy libxml2 yajl
```

On Ubuntu, install:

```
$ sudo apt install -y jq libjson-pp-perl perltidy xmllint libxml2-utils yajl-tools
```

Now we can perform a query:

```
$ curl -sq /port/1/1/1
{
  "candela.lanforge.HttpPort" : {
     "duration" : "1"
  },
  "handler" : "candela.lanforge.HttpPort$JsonResponse",
  "interface" : {
     "stuff":...
  },
  "uri" : "port/:shelf_id/:resource_id/:port_id"
}
```

Notice that the URI object list paths with colon-tagged positions in them, e.g.: `/cli-form/:cmd`. These are interpreted as URL parameters and not query string parameters, they cannot be moved into the query string.

Making your shell friendly

To save you typing, you might want to add this function to your `.bash_aliases` file:

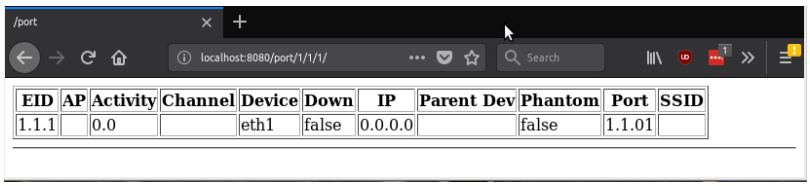
```
function Json() {
   curl -sqv -H 'Accept: application/json' "http://localhost:8080${@}" \
    | json_reformat | less
}
```

Then you can make your calls this way:

```
$ Json /port/1/1/1
```

Browsing results in table format

We can view a URL in a browser as well:



EID	AP	Activity	Channel	Device	Down	IP	Parent Dev	Phantom	Port	SSID
1.1.1		0.0		eth1	false	0.0.0.0		false	1.1.01	

Viewing Alerts and Events

You can both view and stream event data. Querying events and alerts are both quite similar:

```
$ Json /events
{
"handler" : "candela.lanforge.HttpEvents$FixedJsonResponder",
"events" : [
   {
      "2249259" : {
         "event" : "Connect",
         "_links" : "/events/2249259",
         "entity id" : "NA"
      }
   },
   ....
```

A busy LANforge system will generate hundreds of thousands of events. Only the last few thousand can be recalled.

You can inspect a singular event:

```
$ Json /events/2249259
{
    "handler": "candela.lanforge.HttpEvents$FixedJsonResponder",
    "uri": "events/:event_id",
    "candela.lanforge.HttpEvents": {
        "duration": "0"
    },
    "event": {
        "eid": "1.3.21",
        "entity id": "NA",
        "event": "Connect",
        "event description": "sta3106 (phy #1): connected to 00:0e:8e:d5:fa:e6",
        "id": "2249259",
        "name": "sta3106",
        "priority": "  Info",
        "time-stamp": "2018-07-24 14:39:33.776",
        "type": "Port"
    }
}
```

We can view `/alerts` similarly.

```
$ Json /alerts/92
{
   "handler" : "candela.lanforge.HttpEvents$FixedJsonResponder",
   "uri" : "alerts/:event_id",
   "alert" : {
      "name" : "wlan0",
      "time-stamp" : "2018-07-02 16:23:30.880",
      "entity id" : "NA",
      "id" : "92",
      "eid" : "1.1.5",
```

Streaming Events

Continually polling the `/events` URL is not as effective as streaming a websocket providing the same data. We need a web socket client. Websockets are built into modern browsers and there are python and perl utilities for the job as well. An easy to use python client is `wsdump`.

Installing wsdump

There is a useful python utility called `wsdump` (or `wsdump.py`). Try to install the `python-websocket` package to get it. There are many similar matches, but there is not one dedicated package that provides it. On Fedora:

```
root@fedora$ dnf whatprovides `which wsdump`
root@fedora$ dnf install -y python3-websocket-client
```

```
root@ubuntu$ ls -l /usr/bin/wsdump
  /usr/bin/wsdump → /etc/alternatives/wsdump
root@ubuntu$ ls -l /etc/alternatives/wsdump
 /etc/alternatives/wsdump → /usr/bin/python2-wsdump
root@ubuntu$ dpkg-query -S /usr/bin/python2-wsdump
python-websocket: /usr/bin/python2-wsdump

root@ubuntu$ sudo apt install python-websocket
```

You might need to install pip, and that might be in the python3-pip package. Then you can install via:

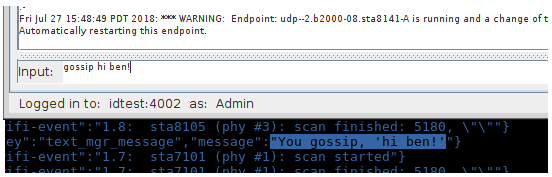
```
$ sudo apt install python-pip # or sudo dnf install python-pip
$ sudo pip install --upgrade pip
$ pip search websocket
$ sudo pip install websocket-client
```

Streaming Using `wsdump`

Here's an example of `wsdump` below. Don't forget you are now using h the `ws://` schema and not the `http://` schema!

```
$ /usr/bin/wsdump ws://localhost:8081/
```

It might take a few second to start showing results if your system is not very active. You should be able to prompt output by executing this message in the **Messages** tab: `gossip hi ben!`



Streaming Using `javascript`

You can also use a web page to follow events because websockets are built into modern browsers. This is a screenshot of the

Data Views

URLs

/shelf

The `/shelf/1/` URL provides a list of resources in your realm:


```
$ Json /shelf/1
{
    "handler": "candela.lanforge.HttpResource$JsonResponse",
    "uri": "shelf/:shelf_id",
    "candela.lanforge.HttpResource": {
        "duration": "0"
    },
    "resources": [
        {
            "1.1": {
                "_links": "/resource/1/1",
                "hostname": "idtest.candelatech.com"
            }
        },
        {
            "1.2": {
                "_links": "/resource/1/2",
                "hostname": "hedtest"
            }
        }
   ]
}
```

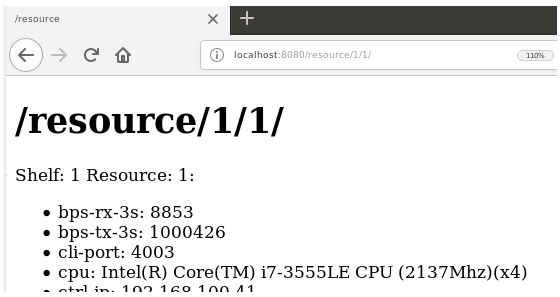
/shelf/1/

Resource 1
Resource 2
Resource 3
Resource 4
Resource 5
Resource 6
Resource 7
Resource 8

/resource

The `/resource` URL provides a digest of ports available at the requested resource.

```
$ Json /resource/1/1
{
   "handler" : "candela.lanforge.HttpResource$JsonResponse",
   "resource" : {
      "free swap" : 526332,
      "free mem" : 4634228,
      "load" : 0.4,
      "bps-rx-3s" : 7850,
      "sw version" : "  5.3.8  64bit",
      "entity id" : "NA",
      "tx bytes" : 40533976395,
      "phantom" : false,
      "eid" : "1.1",
      "hostname" : "idtest.candelatech.com",
      "hw version" : "Linux/x86-64",
```



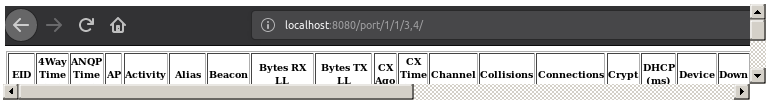
/port

The `/port` URL provides a digest of ports and their state. You can request multiple ports by ID on this resource by appending the port IDs with commas. You can list ports on a resource:

```
$ Json /port/1/5/list
{
   "handler" : "candela.lanforge.HttpPort$JsonResponse",
   "uri" : "port/:shelf_id/:resource_id/:port_id",
   "interfaces" : [
      {
         "1.5.b5000" : {
            "entity id" : "NA",
            "_links" : "/port/1/5/7",
            "alias" : "b5000"
         }
      },
      {
         "1.5.eth0" : {
            "alias" : "eth0",
```

We can query multiple ports at a time by their number or their name by placing a comma between the specifiers. Additionally, we can query for just the fields we desire. All field names are lower-case: `?fields=tx+crr,rx+fifo`.

```
$ Json '/port/1/5/wiphy0,wiphy1?fields=device,phantom,tx+bytes,mode'
{
  "interfaces" : [
     {
        "1.5.wiphy0" : {
           "tx bytes" : 401236186,
           "mode" : "802.11abgn",
           "device" : "wiphy0",
           "phantom" : false
        }
     },
     {
        "1.5.wiphy1" : {
           "phantom" : false,
           "device" : "wiphy1",
```



/cx

The `/cx` URL allows us to query Layer-3 connection information.

```
$ Json /cx
{
   "uri" : "cx",
   "handler" : "candela.lanforge.GenericJsonResponder",
   "connections" : [
      "41.1" : {
         "entity id" : "NA",
         "name" : "udp:r3r2:3000",
         "_links" : "/cx/41"
      },
      "50.1" : {
         "name" : "udp:r3r2:3009",
         "entity id" : "NA",
         "_links" : "/cx/50"
      }
```

And individual connections:

```
$ Json /cx/udp:r3r2:3000$ Json 'cx/udp:r3r2:3000'
{
   "uri" : "cx/:cx_id",
   "41.1" : {
      "drop pkts b" : 0,
      "type" : "LF/UDP",
      "rx drop % a" : 0,
      "rpt timer" : "1000",
      "pkt rx a" : 0,
      "avg rtt" : 0,
      "rx drop % b" : 0,
      "name" : "udp:r3r2:3000",
      "endpoints (a ↔ b)" : "udp:r3r2:3000-A <=> udp:r3r2:3000-B",
      "drop pkts a" : 0,
      "entity id" : "NA",
```

> Technically, colons in URLs need to be encoded as `%3A`, so the above URL should be `/cx/udp%3Ar3r2%3A3000`, but `curl` is pretty darned forgiving.

/endp

Endpoints may be listed and inspected:

```
$ Json /endp/
            {
   "uri" : "endp",
   "handler" : "candela.lanforge.HttpEndp$JsonResponse",
   "candela.lanforge.HttpEndp" : {
      "duration" : "4"
   },
   "endpoint" : [
      {
         "1.2.8.55.2" : {
            "_links" : "/endp/55",
            "entity id" : "NA",
            "name" : "sta3000-ep-B"
         }
      },
```

```
$ Json /endp/sta3000-ep-B
   {
      "candela.lanforge.HttpEndp" : {
         "duration" : "1"
      },
      "uri" : "endp/:endp_id",
      "endpoint" : {
         "rx rate ll" : 0,
         "pdu/s tx" : 0,
         "bursty" : false,
         "rx rate" : 0,
         "tx pkts ll" : 0,
         "rx bytes" : 0,
         "run" : false,
         "tcp rtx" : 0,
```

Creating Ports

It is possible to create ports and connections by using the CLI commands. Your LANforge test scenarios (located in the `/home/lanforge/DB/` directory) contain all the CLI commands that create your ports and connections. You can submit those commands over HTTP in two ways:

- `/cli-json/$command` An example of using the *gossip* command:

  ```
  curl -X POST -H 'Content-type: application/json' \
                -d '{"message":"hello world"}' http://localhost:8080/cli-json/gossip
  ```

  Then check your LANforge GUI messages.

- `/cli-form/$command` An example of using the *gossip* command:

  ```
  curl -X POST -d 'message=hello+world' http://localhost:8080/cli/gossip
  ```

  Then check your LANforge GUI messages.

- `/cli/`: use this method to submit a raw URL-encoded command. This might be useful if you are copying commands directly out of a database:

  ```
  curl -X POST -d 'cmd=gossip hello' http://localhost:8080/cli/
  ```

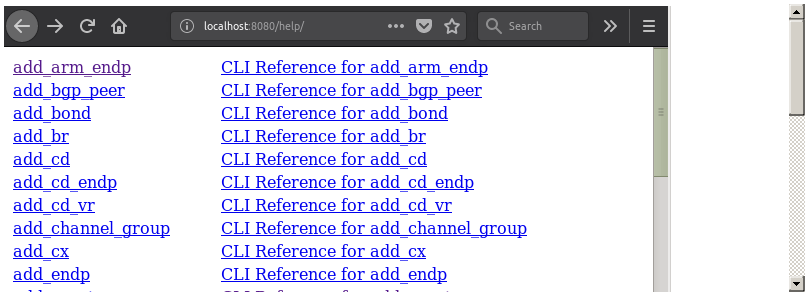
Except for `/cli-json`, these methods accept `application/x-www-form-urlencoded` content type submissions. This is default for the NanoHttp library and default for `curl`.

These CLI commands do not return data, only a result code. All data that the Perl scripts would collect from command line queries is sent directly to the GUI. Some CLI commands send data over the websocket, like the `diag` command.

Command help

Commands are often complex and include a number of bitwise flags to set the state and features of ports. There is presently no tag-substitution for port flags, but there is a help utility that can help you compute them.

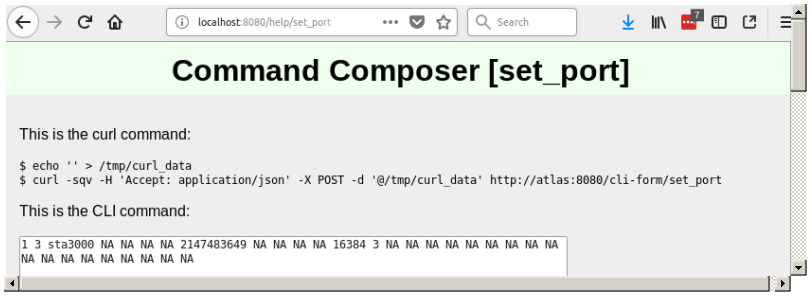
```
http://localhost:8080/help/
```



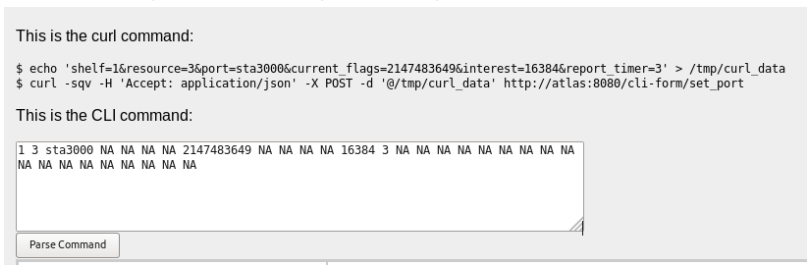
Select a command to see the field helper screen:

```
http://localhost:8080/help/set_port
```

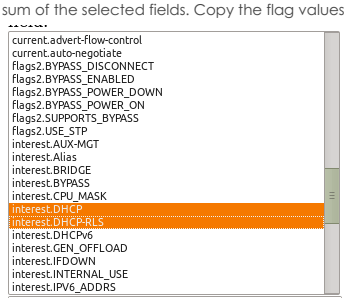
Type values into the field inputs and the CLI command will be refreshed:



Click the **Parse Command** button and the values in the command box will be displayed in the curl command and the field inputs. (Notice this form is doing a GET request.)



You may find a list of flag fields that are organized by field names. The text area below the selection list is the sum of the selected fields. Copy the flag values into the input field above to incorporate it into your command.



Creating a WiFi Station

Please refer to the scripts `lf_associate_ap.pl` and `lf_vue_mod.sh` for examples of how to produce lists of CLI commands involved in creating stations. Please refer to:

1. Learn CLI Commands used to operate WiFi stations
2. and Changing Station WiFi SSID with the CLI API

These will provide ways of collecting the CLI commands in log files for you to place into the command `/help/` page.

- Use ssh to log into your LANforge manager. Use the `lf_vue_mod.sh` script to create a station:

```
$ cd scripts
$ ./lf_vue_mod.sh --mgr localhost --resource 3 --create_sta --name sta3101 \
  --radio wiphy1 --ssid idtest-1000-open --passphrase '[BLANK]' \
  --log_cli /tmp/clilog.txt

$ cat /tmp/clilog.txt
set_wifi_radio 1 3 wiphy1 NA -1 NA NA NA NA NA NA NA NA 0x1 NA
add_sta 1 3 wiphy1 sta3101 1024 idtest-1000-open NA [BLANK] AUTO NA 00:0e:8e:c1:df:45 8 NA NA
set_port 1 3 sta3101 0.0.0.0 255.255.0.0 0.0.0.0 NA 2147483648 00:0e:8e:c1:df:45 NA NA NA 8405
```

1. Enter each command into the your browser toolbar by altering the command into a url:

```
http://localhost:8080/help/set_wifi_radio?cli=1 3 wiphy1 NA -1 NA NA NA NA NA NA
```

Produces:

```
$ echo 'shelf=1&resource=3&radio=wiphy1&channel=-1&flags=0x1' > /tmp/curl_data
$ curl -sqv -H 'Accept: application/json' -X POST -d '@/tmp/curl_data' \
 http://localhost:8080/cli-form/set_wifi_radio
```

```
http://localhost:8080/help/add_sta?cli=1 3 wiphy1 sta3101 1024 idtest-1000-open
```

Produces:

```
$ echo 'shelf=1&resource=3&radio=wiphy1&sta_name=sta3101&flags=1024&ssid=idtest-
$ curl -sqv -H 'Accept: application/json' -X POST -d '@/tmp/curl_data' \
 http://localhost:8080/cli-form/add_sta
```

```
http://localhost:8080/help/set_port?cli=1 3 sta3101 0.0.0.0 255.255.0.0 0.0.0.0
```

Produces:

```
$ echo 'shelf=1&resource=3&port=sta3101&ip_addr=0.0.0.0&netmask=255.255.0.0&gate
$ curl -sqv -H 'Accept: application/json' -X POST -d '@/tmp/curl_data' \
 http://localhost:8080/cli-form/set_port
```

2. Verify with the LANforge GUI the changes you wish to make.

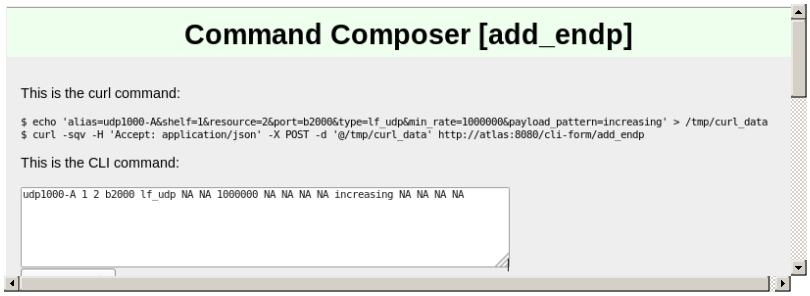
Creating Connections

Using the `/cli-json/add_endp` and `/cli-json/add_cx` URLs, it is possible to create Layer-3 connections. Create the Layer-3 endpoints first, of course.

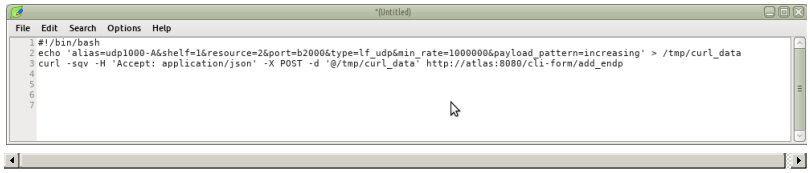
Create L3 Endpoints

Construct your command using the `/help/add_endp` page. For an example, use these parameters:

alias	enter udp1000-A
shelf	1
resource	2
port	b2000
type	select **type.lf_udp**
min_rate	1000000 (1 Mbps)
max_rate	SAME
payload_pattern	select **payload_pattern.increasing**

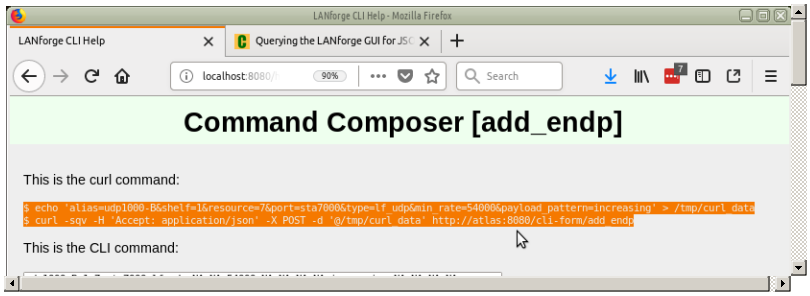


Click **Parse Command** and copy the resulting `curl` command into a text editor:

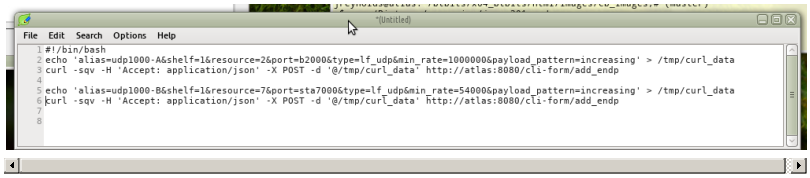


And for the **B** endpoint, choose a station:

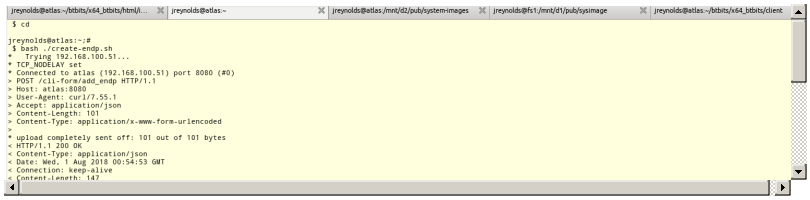
alias	enter udp1000-B
shelf	1
resource	7
port	sta7000
type	select **type.lf_udp**
min_rate	54000 (54 Kbps)
max_rate	SAME
payload_pattern	select **payload_pattern.increasing**



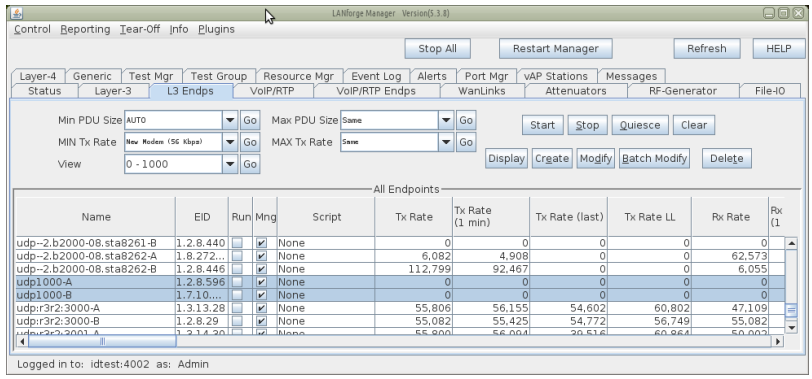
Click **Parse Command** and copy the resulting `curl` command into a text editor:



We'll save this file as a shell script: `~/create-endp.sh` We can then run it from our terminal like so: `bash -x create-endp.sh`

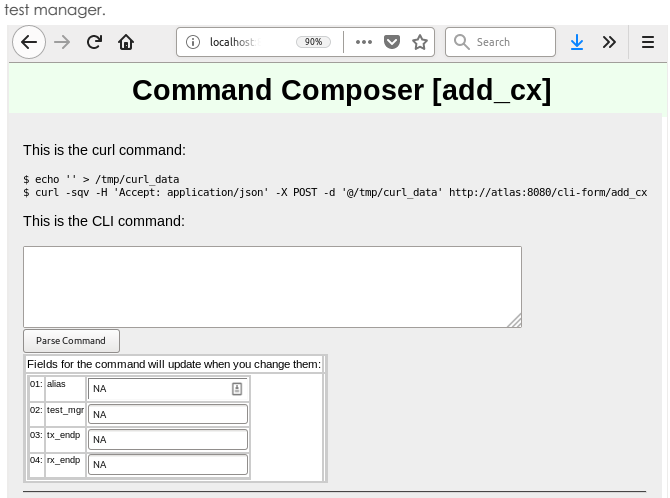


We should see the endpoints we've created in the LANforge GUI**Endps** tab:



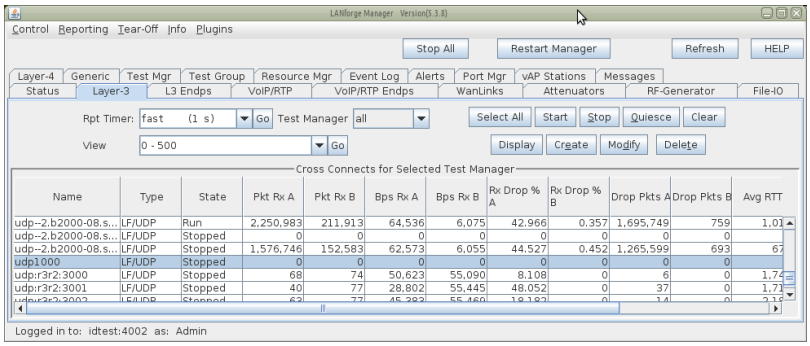
Create L3 Connection

With the creation of two endpoints, we can proceed with creating a Layer 3 cross-connect. This is much simpler, it really only takes the names of the two endpoints we created above. We'll choose `default_tm` for the test manager.



alias	udp1000
test_mgr	default_tm
tx_endp	udp1000-A
rx_endp	udp1000-B

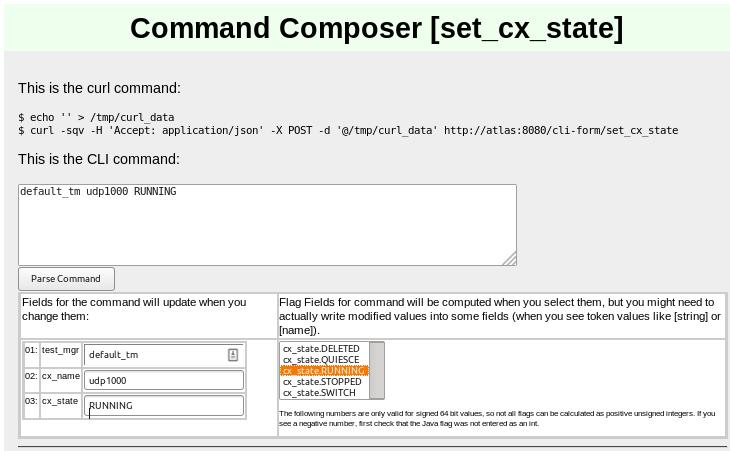
Click the **Parse Command** button and copy the resulting `curl` command into your editor with the shell script. Run the script again. It doesn't hurt to re-create the endpoints.



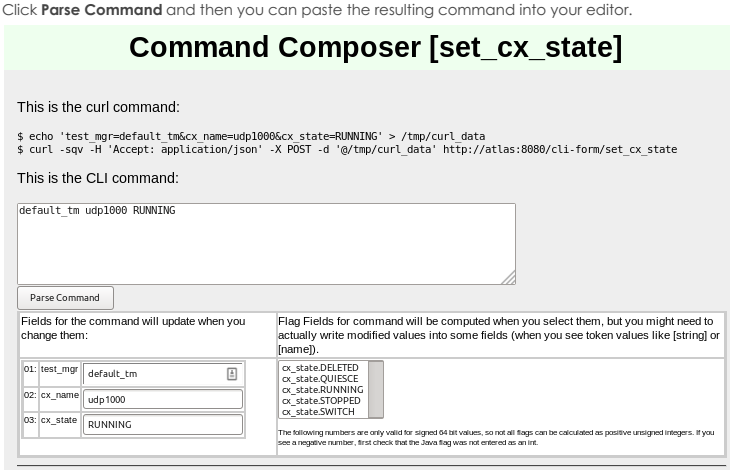
Toggling the Connection

Cross connects have three good state: STOPPED, RUNNING, and QUIESCE. The command to change them is `set_cx_state`. You will have no trouble creating the command:

test_mgr	default_tm
cx_name	udp1000
cx_state	RUNNING



Click **Parse Command** and then you can paste the resulting command into your editor.



Advanced Techniques

You can make JSON submissions and you can also submit Base64 encoded values in both form an and JSON submission URLs.

Submitting Base64

Field names that end in `-64` are interpreted as base64 encoded values. From a linux terminal, you can convert text to base64 encoded value using the `base64` command:

```
$ echo "RUNNING" | base 64
UlVOTklORwo=
```

Below is a CLI command example. **You typically would not care to spend the effort doing this unless the data you need to express is difficult to URL encode.**

```
$ echo 'test_mgr-64=YW55Cg==&cx_name-64=dWRwMTAwMAo=&cx_state-64=UlVOTklORwo=' > /tmp/curl_data
$ curl -A 'Accept: application/json' -X POST -d @/tmp/curl_data http://host/cli-form?set_cx_state
```

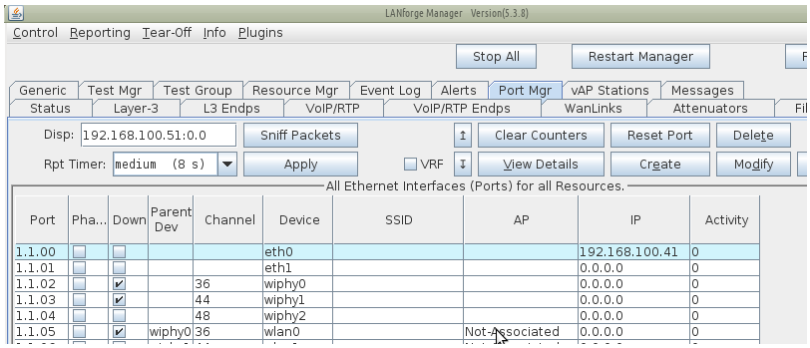
Submitting JSON

Instead of posting to `/cli-form`, you can post to `/cli-json` and your submission will be parsed as a json object. The parameter names stay the same. The base64 name extensions are also available! You **need** to specify that your `Content-type` in the POST is `application/json`.

```
$ echo '{"test_mgr":"default_tm","cx_name":"udp1000","cx_state":"RUNNING"}' > /tmp/curl_data
$ curl -sq -H 'Content-type: application-json' -H 'Accept: application/json' \
  -X POST -d@/tmp/curl_data http://localhost:8080/cli-json/set_cx_state
```

Handling Mismatched Column Errors

(This should be fixed as of 2018/08/14) When the LANforge cliet is in GUI mode, the **columns** of data that are returned match the GUI **table columns** displayed. You can use the *Right-click→Add/Remove Table Columns* menu item to change this. **We do not recommend doing this** for querying JSON data though, because the table columns definitions will not match up to the data the webserver expects to return.



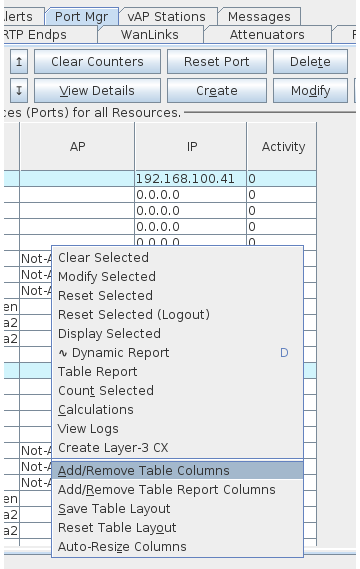

```
$ curl -sq http://localhost:8080/port | json_pp
{
   "error_list" : [
      "names_to_col_ids map is not going to work:\n[tx abort][25 > 4]\n[cx time (us)][48 > 4]
   ],
   "status" : "INTERNAL_ERROR"
}
```

The terminal you started the LANforge client on will also give a similar error:

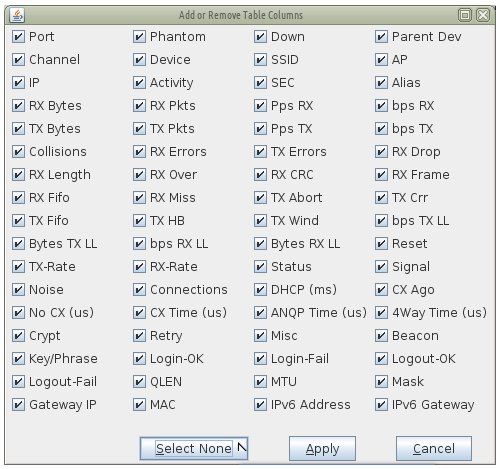
```
1532480073953:  names_to_col_ids size:71
java.lang.IllegalArgumentException: names_to_col_ids map is not going to work:
1532480073953:  lfj_table columns:10
```

Reset the Table Layout

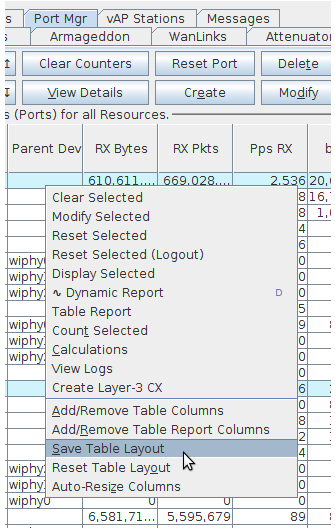
1. Right-clicking the Port Mgr and selecting **Add/Remove Table Columns** will allow you to change this.



2. Clicking the **Select All/None** button and then **Apply** will get all the columns displayed, and returned in your queries.



3. Make sure to **Right-Click → Save Table Layout** so that your next session will show all the data.



4. Restart the LANforge client

Learn CLI commands used to operate WiFi stations.

Goal: Compare and learn script and CLI commands used when creating and operating stations.

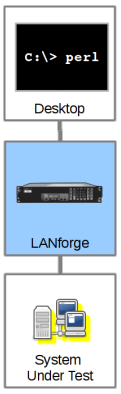
The LANforge perl scripts have always been able to print out the CLI commands used to communicate with the LANforger manager. These examples show recent modifications that allow you to collect the CLI commands more easily using the `--log_cli` switch. (Not all scripts have this feature yet). The `--log_cli` switch prints CLI commands to your console. You can collect those commands in a file using the switch with an argument of the file name:

```
$ ./lf_vue_mod.sh --log_cli /tmp/clilog.txt
```

It is possible to repeat these commands using the `lf_firemod.pl` script:

```
$ ./lf_firemod.pl --mgr localhost --action do_cmd \
--cmd "set_port 1 2 sta200 NA NA NA NA 0 NA NA NA NA 8388610"
```

Requires a LANforge CT520 (or better) system and an access point.



Examples of CLI commands

1. Creating Stations
2. Using Open Authentication
3. Using WPA2 Authentication
4. Static IP Addresses
5. Station DHCP IP Address
6. Creating a Station with a MAC Address Pattern
7. Admin Down
8. Admin Up
9. Deleting a Station
10. Creating Connections and Running Traffic
11. Starting and Stopping Traffic
12. Create a Layer 3 TCP Connection
13. Create a Layer 3 UDP Connection
14. Create a Layer 4-7 Web Connection

Setting for Examples

This was done in a two-machine LANforge cluster, the manager named *jedtest* and the second resource named *kedtest*. The CLI output of these CLI commands has been discarded as well as any `show_port` commands.

The `show_port` commands are useful for inspecting the results of previous commands. Often there is useful wait before issuing the `show_port` command to allow processing time on the manager. Please inspect the scripts in the `/home/lanforge/scripts` directory for how and when they tend to sleep.

These commands are also found in the `/home/lanforge/DB/DFLT` directory files. You cannot run those DB files directly, because they are executed in certain order. However, you can grep for connection- and station-names in those files to find results of GUI commands.

Creating Stations

Using Open Authentication

This station is created with DHCP enabled. That is controlled via *flags* that are descibed in the add_sta command.

Shell script:

```
./lf_vue_mod.sh --mgr jedtest --resource 2 --create_sta --name sta100 --radio wiphy0 --security open --ssid jedtest
```

Perl script:

```
./lf_associate_ap.pl --mgr jedtest --resource 2 --quiet yes --action add --radio wiphy0 --security open --ssid jedtest --passphrase --first_sta sta100 --first_ip DHCP --num_stations 1
```

CLI command:

```
set_wifi_radio 1 2 wiphy0 NA -1 NA NA NA NA NA NA NA NA 0x1 NA
add_sta 1 2 wiphy0 sta100 0 jedtest NA [BLANK] AUTO NA 00:0e:8e:8d:8d:e9 8 NA NA NA
set_port 1 2 sta100 0.0.0.0 255.255.0.0 0.0.0.0 NA 2147483648 00:0e:8e:8d:8d:e9 NA
```

Using WPA2 Authentication

This station is created with DHCP enabled. That is controlled via *flags* that are descibed in the add_sta command.

Shell script:

```
$ ./lf_vue_mod.sh --mgr jedtest --resource 2 --create_sta --name sta200 \
 --radio wiphy1 --security wpa2 --ssid jedtest --passphrase jedtest1 \
 --log_cli /tmp/clilog.txt
```

Perl script:

```
./lf_associate_ap.pl --mgr jedtest --resource 2 \
  --action add --radio wiphy1 --security wpa2 --ssid jedtest \
  --passphrase jedtest1 --first_sta sta200 --first_ip DHCP --num_stations 1
```

CLI command:

```
set_wifi_radio 1 2 wiphy1 NA -1 NA NA NA NA NA NA NA NA 0x1 NA
add_sta 1 2 wiphy1 sta200 1024 jedtest NA jedtest1 AUTO NA 00:0e:8e:6f:01:62 8 NA
set_port 1 2 sta200 0.0.0.0 255.255.0.0 0.0.0.0 NA 2147483648 00:0e:8e:6f:01:62 NA
```

Static IP Addresses

Here is an example of creating a virtual station with a static address: 10.26.2.14/255.255.254.0

Shell Script:

--

Perl Script:

```
./lf_associate_ap.pl --mgr jedtest --resource 2 --action add --radio wiphy1 --first_sta sta203 --first_ip 10.26.2.4 --netmask 255.255.254.0 --ssid jedtest --security wpa2 --passphrase jedtest1 --num_stations 1 --wifi_mode abgnAC --log_cli /tmp/clilog.txt
```

CLI Command:

```
set_wifi_radio 1 2 wiphy1 NA -1 NA NA NA NA NA NA NA NA 0x1 NA
show_port 1 2 wiphy1
add_sta 1 2 wiphy1 sta203 1024 jedtest NA jedtest1 AUTO NA 00:0e:8e:63:50:62 8 NA
set_port 1 2 sta100 10.26.2.4 255.255.254.0 0.0.0.0 NA 0 00:0e:8e:63:50:62 NA NA NA
```

Station DHCP IP Address

For the station to gain a DHCP IP address, you have to *admin-up* the station.

Creating a Station with a MAC Address Pattern

The `lf_associate_ap` script contains logic that parses a MAC address pattern and produces new MAC addresses. This is not a feature of the LANforge Manager. Your CLI calls to the LANforge manager will not parse the mask.

The pattern nomenclature of the LANforge GUI can also be used when specifying a MAC address for stations:

`xx`
keep parent radio octet

`*`
randomize this octet

`00 - ff`
assign this value to the octet

Shell script:

--

Perl script:

```
./lf_associate_ap.pl --mgr jedtest --resource 2 --action add --radio wiphy1 --first_sta sta205 --first_ip 10.26.2.4 --netmask 255.255.254.0 --ssid jedtest --security wpa2 --passphrase jedtest1 --num_stations 1 --mac-pattern '4e:xx:xx:xx:*:01' --log_cli /tmp/clilog.txt
```

CLI command:

```
set_wifi_radio 1 2 wiphy1 NA -1 NA NA NA NA NA NA NA NA 0x1 NA
show_port 1 2 wiphy1
add_sta 1 2 wiphy1 sta205 1024 jedtest NA jedtest1 AUTO NA 4e:0e:8e:43:f1:01 8 NA
set_port 1 2 sta205 10.26.2.4 255.255.254.0 0.0.0.0 NA 0 4e:0e:8e:43:f1:01 NA NA NA
```

Admin Down

Shell script:

```
./lf_vue_mod.sh --mgr jedtest --resource 2 --down --name sta200 --log_cli /tmp/clilog.txt
--quiet 1
```

Perl script:

```
./lf_portmod.pl --manager jedtest --card 2 --port_name sta200 --set_ifstate down
```

CLI command:

```
set_port 1 2 sta200 NA NA NA NA 1 NA NA NA NA 8388610
```

Admin Up

Shell script:

```
./lf_vue_mod.sh --mgr jedtest --resource 2 --up --name sta200 --log_cli /tmp/clilog.txt --
quiet 1
```

Perl script:

```
./lf_portmod.pl --manager jedtest --card 2 --port_name sta200 --set_ifstate up
```

CLI command:

```
set_port 1 2 sta200 NA NA NA NA 0 NA NA NA NA 8388610
```

Delete Station

Shell script:

```
./lf_vue_mod.sh --mgr jedtest --resource 2 --delete_sta --name sta200 --log_cli
/tmp/clilog.txt --quiet 1
```

Perl script:

```
./lf_associate_ap.pl --mgr jedtest --resource 2 --action del --port_del sta200
```

CLI command:

```
rm_vlan 1 2 sta100
```

Creating Connections and Running Traffic

LANforge can create Layer-3 and Layer 4-7 connections using the `lf_vue_mod.sh` script. When connections are created, they exist in a *stopped* state. Connections can then have their state changed to *RUNNING* to start traffic.

Starting and Stopping Traffic

Layer-3 and Layer 4-7 connections both subject to the states STOPPED, RUNNING, and QUIECSE.

Shell script:

```
./lf_vue_mod.sh --mgr jedtest --start_cx --name tcp200 --log_cli /tmp/clilog.txt --quiet 1
```

```
./lf_vue_mod.sh --mgr jedtest --stop_cx --name tcp200 --log_cli /tmp/clilog.txt --quiet 1
```

Perl script:

```
./lf_firemod.pl --mgr jedtest --action do_cmd --cmd "set_cx_state default_tm tcp200 RUNNING"
```

```
./lf_firemod.pl --mgr jedtest --action do_cmd --cmd "set_cx_state default_tm tcp200 STOPPED"
```

CLI commands:

```
set_cx_state default_tm tcp200 RUNNING
```

```
set_cx_state default_tm tcp200 STOPPED
```

Create a Layer 3 TCP Connection

Shell script:

```
./lf_vue_mod.sh --mgr jedtest --resource 2 --create_cx --name tcp200 --tcp --sta sta200 --
port eth1 --bps 1000000 --log_cli /tmp/clilog.txt --quiet 1
```

Perl script:

```
./lf_firemod.pl --mgr jedtest --resource 2 --action create_endp --endp_name tcp200-
./lf_firemod.pl --mgr jedtest --resource 2 --action create_endp --endp_name tcp200-
./lf_firemod.pl --mgr jedtest --resource 2 --action create_cx --cx_name tcp200 --cx
```

CLI commands:

```
add_endp tcp200-A 1 2 sta200 lf_tcp -1 NA 1000000 1000000 NA -1 -1 increasing NO NA 0 0
set_endp_report_timer tcp200-A 5000
add_endp tcp200-B 1 2 eth1 lf_tcp -1 NA 1000000 1000000 NA -1 -1 increasing NO NA 0 0
set_endp_report_timer tcp200-B 5000
add_cx tcp200 default_tm tcp200-A tcp200-B
set_cx_report_timer default_tm tcp200 5000 NA
```

Create a Layer 3 UDP Connection

Shell script:

```
./lf_vue_mod.sh --mgr jedtest --resource 2 --create_cx --name upd200 --udp --sta sta200 --
port eth1 --bps 2000000 --log_cli /tmp/clilog.txt --quiet 1
```

Perl script:

```
./lf_firemod.pl --mgr jedtest --resource 2 --log_cli /tmp/clilog.txt --quiet 1 --ac
./lf_firemod.pl --mgr jedtest --resource 2 --log_cli /tmp/clilog.txt --quiet 1 --ac
./lf_firemod.pl --mgr jedtest --resource 2 --log_cli /tmp/clilog.txt --quiet 1 --ac
```

CLI commands:

```
add_endp udp200-A 1 2 sta200 lf_udp -1 NA 2000000 2000000 NA -1 -1 increasing NO NA 0 0
set_endp_report_timer udp200-A 5000
add_endp udp200-B 1 2 eth1 lf_udp -1 NA 2000000 2000000 NA -1 -1 increasing NO NA 0 0
set_endp_report_timer udp200-B 5000
add_cx udp200 default_tm udp200-A udp200-B
```

```
set_cx_report_timer default_tm udp200 5000 NA
```

Create a Layer 4-7 Web Connection

Layer 4-7 connections are created with a one-sided technique, the curl command always operates on the**A-side** and the **B-side** is unmanaged. The endpoint and connection naming does not follow the Layer-3 convention.

Shell script:

```
./lf_vue_mod.sh --mgr jedtest --resource 2 --create_l4 --name yh200 --sta sta200 --url http://www.yahoo.com/ --utm 2400 --log_cli /tmp/clilog.txt --quiet 1
```

Perl script:

Commands are set using `lf_firemod.pl --action do_cmd --cmd ...`

CLI commands:

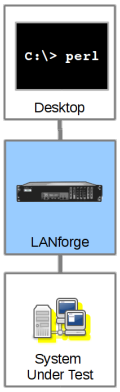
```
add_l4_endp yh200 1 2 sta200 l4_generic 0 10000 2400 'dl http://www.yahoo.com/ /dev
set_endp_tos yh200 DONT-SET 0
set_endp_flag yh200 L4Enable404 0
set_endp_report_timer yh200 5000
set_endp_flag yh200 ClearPortOnStart 0
set_endp_quiesce yh200 3
add_cx CX_yh200 default_tm yh200
```

Learn CLI commands used create Generic endpoints.

Goal: Compare and learn script and CLI commands used when creating and operating Generic endpoints.

Similar to the Layer3 perl script, `lf_firemod.pl`, the `lf_generic_ping.pl` script has been enhanced to use curl or other commands with parameter expansions. The `lf_curl.sh` script is a helper script that wraps curl commands and reports success or failure.

Introduced in LANforge 5.3.8.



Example Commands

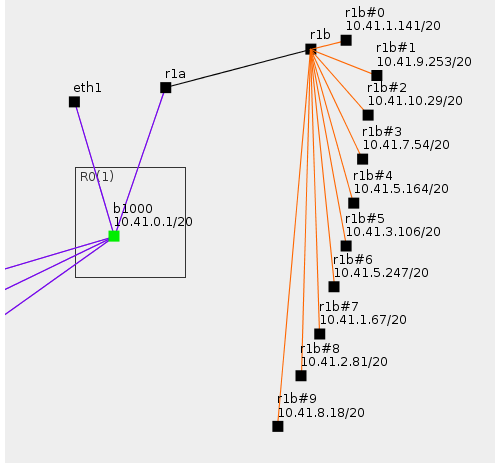
1. Creating ports and MAC VLANs
2. Creating ping endpoints
3. Creating curl endpoints
4. Using parameters for endpoint options
5. Using the `lf_curl.sh`script

Setting for Examples

Generic endpoints are effectively one-legged connections. You can ping an IP or use curl to download web content. Both of these types of connections can be pointed back at the LANforge itself. We can operate these connections from redirect interfaces. The same techniques can apply to WiFi stations, of course.

Create Redirects

1. Create a bridge `br0` including `eth1`
2. In Netsmith, select *br0→Modify* and enable DHCP
3. Create a redirect `r1a, r1b`
4. Add `r1a` to `br0`
5. Create 10 MAC VLANs based off `r1b` with DHCP enabled



6. After the MAC VLAN ports have addresses, you can verify that you can ping and download pages from LANforge webserver:
   1. In a terminal, begin by sourcing `lanforge.profile`:

      ```
      /home/lanforge$ . lanforge.profile
      ```

   2. Ping the bridge from a MAC VLAN:

      ```
      /home/lanforge$ ping -I 10.41.1.141 10.41.0.1
      ```

   3. Grab the web page. (Apache is listening on all ports by default.)

      ```
      /home/lanforge$ curl -sq --interface 10.41.1.141 http://10.41.0.1/
      ```

Creating ping endpoints

In the `/home/lanforge/scripts` directory, the `lf_generic_ping.pl` script creates a wrapped ping command by default. There is another script, `lfping`, that reports ping results to LANforge. Here is an example of creating 1 ping endpoint:

```
/home/lanforge/scripts$ ./lf_generic_ping.pl --mgr localhost --resource 1 \
     --dest 10.41.0.1 --interface r1b#0 --name pingtest
```

Create a ping endpoint for every MAC VLAN parented by `r1b`:

```
$ ./lf_generic_ping.pl --mgr localhost --resource 1 \
    --dest 10.41.0.1 --parent r1b
```

Create a ping endpoint for every virtual station parented by `wiphy0`:

```
$ ./lf_generic_ping.pl --mgr localhost --resource 1 \
    --dest 10.41.0.1 --radio wiphy0
```

Create a ping endpoint for every MAC VLAN beginning with `r1b#1` (r1b#1, r1b#10):

```
$ ./lf_generic_ping.pl --mgr localhost --resource 1 \
    --dest 10.41.0.1 --match 'r1b#1'
```

Creating curl endpoints

To use other commands with the script, you can create a `--cmd` parameter. You can use curl directly if desired, but curl's output is not formatted well for LANforge to understand. By default, commands do not understand what port or IP they should be interacting as. We need to provide special parameters to help.

Parameter expansion

The `lf_generic_ping.pl` script will look for these tokens in the `--cmd` parameter:

- `%i` Expands to the IPv4 address of the port.
- `%d` Expands to the destination hostname or address
- `%p` Expands to the port name

The curl wrapper script

The `scripts/lf_curl.sh` script is a wrapper for curl that detects success or failure, and an operate the request in a loop. Expandable parameters are expanded by lf_generic_ping.pl, not lf_curl.sh.

You can use lf_curl.sh from the command line to test it out:

```
$ ./lf_curl.sh -i 10.41.1.141 -p r1b#1 -o /tmp/output -d http://example.com/
```

Executes:

```
curl -sqLk --interface 10.0.0.1 -o /tmp/output_r1b#1 http://example.com/
```

So it is best used from lf_generic_ping.pl to construct commands referencing this script:

```
./lf_generic_ping.pl --mgr localhost --resource 1 \
     --name curl_ex_ --match 'r1b#'  --dest http://10.41.0.1/ \
     --cmd 'lf_curl.sh -o /tmp/curl_%p.out -i %i -d %d -p %p'
```

Uploading files with curl

It is possible to use the `lf_generic_ping.pl` script to create URL encoded form posts for uploading files.

1. If you don't have a file to upload, create one here:

```
# cd /var/www/html
```

```
# echo 'file=' > data-2m.asc
```

```
# base64 < data_slug_2048k.bin >> data-2m.asc
```

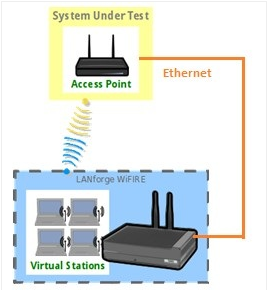
2. Create a series of Generic endpoints:

```
cd /home/lanforge/scripts
./lf_generic_ping.pl --name up \
  --match 'sta' \
  --dest http://192.168.48.1/ \
  --cmd 'curl -d @/var/www/html/data-2m.asc -o /tmp/curl_%p.out --interface %p --dns-ipv4-addr %i %d'
```

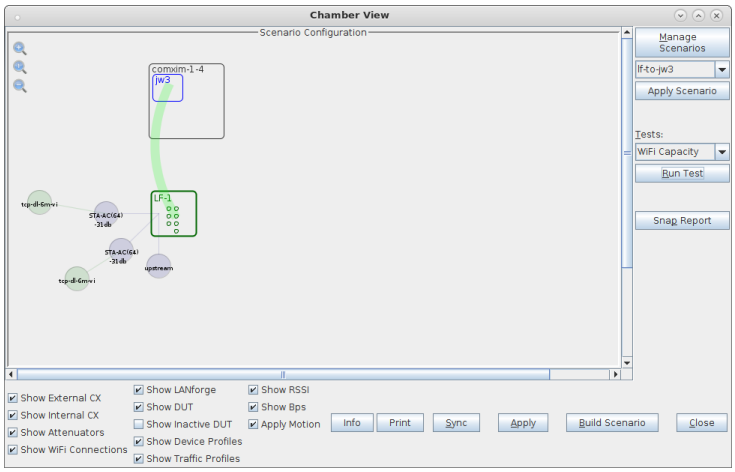
Automate WiFi Capacity and other GUI tests.

Goal: Use a command-line script to have the LANforge-GUI run the WiFi Capacity test and generate a pdf automatically.

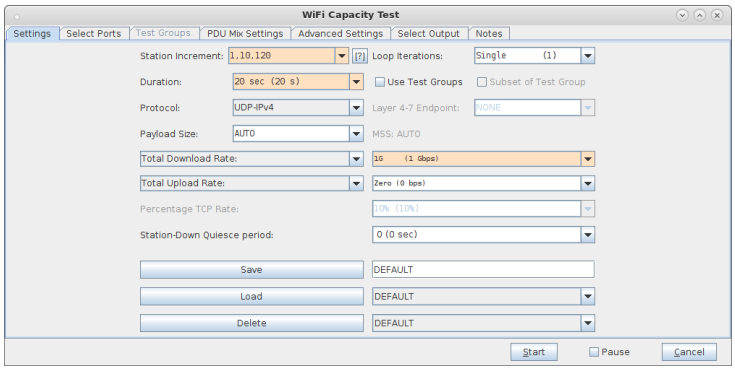
In this test scenario, a script is used to bring up the WiFi Capacity test with a pre-configured configuration. The capacity test is then started and a report is generated. All of this is automated, and other tests such as Dataplane are also supported. This feature requires LANforge version 5.4.1 or higher.



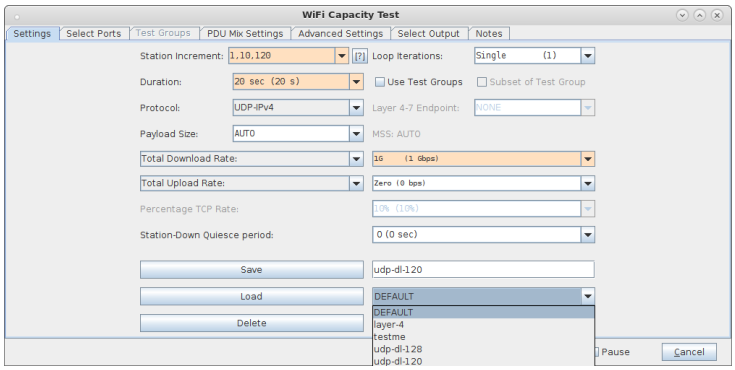
1. Configure WiFi Capacity Test for automated run.
   A. For this to work, the LANforge GUI must be started with the **-cli-socket 3990** argument. This causes it to open a socket to listen for text commands.
   B. Open Chamber View by clicking on the 'Chamber View' button in the LANforge-GUI. Create an appropriate scenario and DUT if you have not already done so. Other cookbook examples have more details of how to do this, please see those if you are unfamiliar with Chamber View.



   C. Select WiFi Capacity test, and click **Run Test** to configure it as desired.

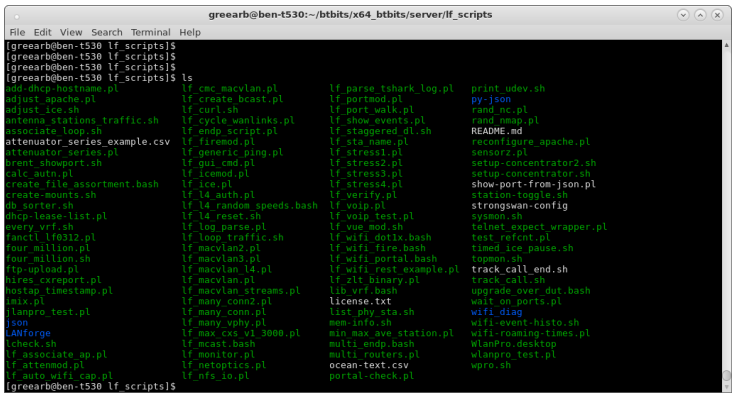


D. Enter a name in the 'Save' field, click save, and make sure it shows up as a loadable configuration. In this case, we are saving the configuration as 'udp-dl-120'

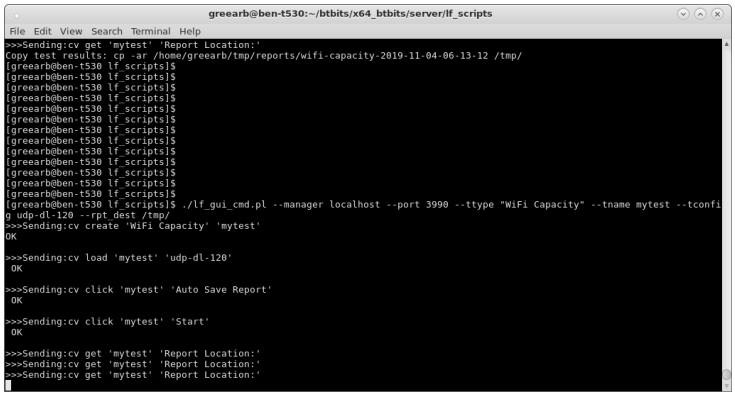


2. Use the lf_gui_cmd.pl script to launch the WiFi Capacity Test.

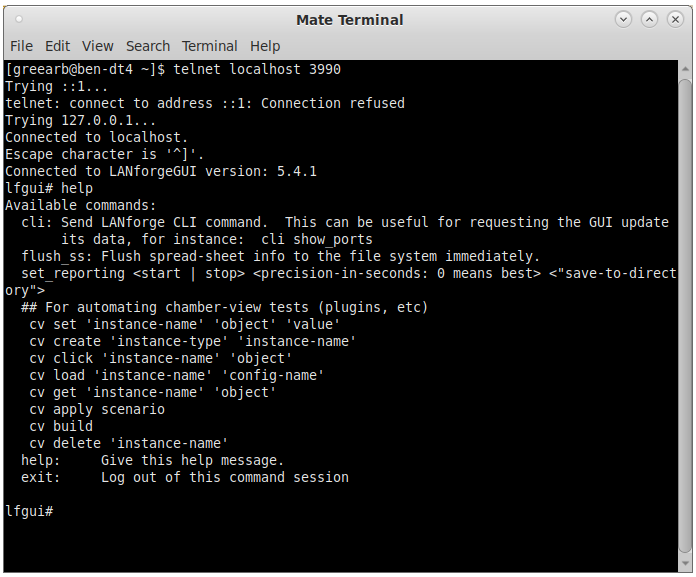
A. Open an ssh session or terminal window and log into the LANforge system, or some other system with the LANforge scripts/ repository. On a LANforge system, this will usually be /home/lanforge/scripts In this case, the directory name is called lf_scripts



B. Run the lf_gui_cli_cmd.pl script with appropriate arguments. Use --help for details. Once you run this, the WiFi Capacity test should be automatically opened and the test will be started. The script will end when the capacity test has completed. You may copy the results to some easily found location, such as a web server directory.



C. For details on what GUI-CLI commands are supported, please see the screen-shot below and look at the contents of the lf_gui_cmd.pl script.



Getting Started with Python scripting for LANforge

Goal: After reading this, a user will know what python modules which are available to use LANforge.

There are many python modules which control LANforge. These modules allow the user to automate many tasks. This cookbook will introduce the initial steps to start LANforge automation, the libraries that need to be imported to run Candela's python scripts, and how to create objects in python.

There are two options to run lanforge_scripts. If you are a programmer, you can clone the git repository locally by using `git clone https://github.com/greearb/lanforge-scripts` in your command line into the directory which the folder should be in.

> i Candela supports Fedora 27+ and Python 3.7+. Older python versions are not supported.

The advantage to cloning lanforge_scripts from git is that it is the latest code available, while the pip repository will lag slightly behind the git repo. The advantage to using pip is that there might be occasional errors in the git repository, which will be ironed out by the time we push it to the pip repository. The pip repository is always close to the master branch of the git repo, lagging no more than a couple weeks. Pick the one which best suits your needs.

> i Please run Python as the **LANforge** user on your system.

If you run python as root, you might break your LANforge system. The LANforge Linux OS requires some certain python packages and versions to operate and configure networking correctly. Candelatech does not recommend updating python packages as root. Your package manager will automatically update these dependencies when you run updates as well, which can overwrite your changes.

The safe way to run python scripts is as a non-root user. If you have python dependencies installed locally by using `pip --user`, if a dependency breaks you can fix it without potentially harming files your operating system needs to operate. Broken python dependencies on your system Python can break your operating system. When your system updates, it might overwrite the changes you have made.

Users are responsible for making certain the version of Python running on their system is supported by the Python Foundation. An up to date list can be found at Python Foundation support. Candela Technologies does not support versions of Python which have been deprecated by the Python Foundation. Customers with a support contract can contact support@candelatech.com to get support upgrading their LANforge systems.

1. Importing libraries:

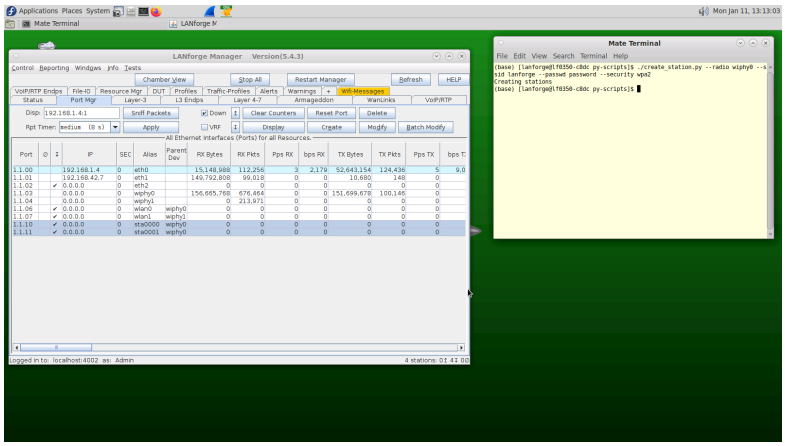
   Please visit the *readme.md* page in *lanforge-scripts/py-scripts* to make sure all the libraries that are needed are imported.

2. Install necessary dependencies
   A. In the root of lanforge_scripts, run `pip3 install --user -r requirements.txt --upgrade`

3. How to run a script from a git clone
   A. We start in the /home/lanforge/lanforge-scripts/py-scripts directory on your lanforge system
   B. On a Linux system, open your terminal and navigate to the py-scripts folder. You can run any python script by typing
      `python3 your_script_here.py --your_flag`
      into your command line. After the script, put any flags you need in order to run your test.

4. How to run a script from pypi
   A. In your terminal, run `python3`
   B. In your python environment, run `import lanforge_scripts`
   C. You can use dir(lanforge_scripts) to see all of the different classes you can use in lanforge_scripts. This is useful if you want to make your own driver script which uses lanforge_scripts as a dependency.

5. Station Creation



   A. The create_station.py script creates wi-fi stations from your terminal.
   B. Type
      `./create_station.py --radio wiphy0 --ssid lanforge --passwd password --security wpa2`
      into your terminal. Please reemember to change the *SSID, Passwd*, and *Security* fields to match your network credentials.
   C. This will create 2 stations on your lanforge device off of your wiphy0 radio. You might need to wait 30 seconds for them to stop being phantom ports.
   D. You can specify the following flags as well:
      A. num_stations - Specify a different number of stations to create off of your antenna. The default is 2.
      B. debug - call this flag if you want detailed diagnostic information

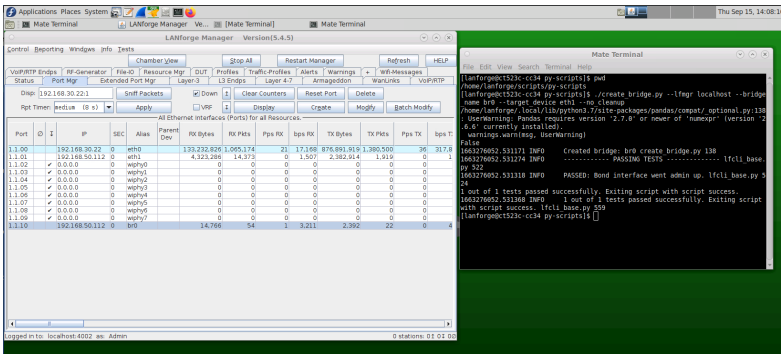
6. Find available networks
   A. You can find available WiFi networks on any Linux device by typing `sudo iw dev wlan0 scan | grep SSID` into your command line
   B. You can also find available WiFi networks by clicking on the station you created in the previous step,

7. Associate to a specific BSSID:

   If you want to connect to a specific MAC Address for your router (which is called the BSSID) you can specify that on each station, you can do that in both the GUI and in the command line.

   A. Connect via Command line
      A. Each of your scripts has an optional AP tag where you can define the MAC address of the router you want to connect to. To do so, simply append `--ap` to the end of the command line argument you are running followed by the router's MAC Address.
   B. Connect via GUI
      A. When you double click on a Station there is an AP field inside WiFi Settings. Type the MAC Address of your router in that box and everything will work assuming the MAC Address is correct.

8. Bridge Creation



A. To create a bridge, you can use the create_bridge.py script in the py_scripts folder.

B. Type

```
./create_bridge.py --lf_mgr localhost --bridge_name br0 --target_device eth1 --no_cleanup
```

Into your terminal, remembering to change the ssid, passwd, and security fields to match your network credentials.

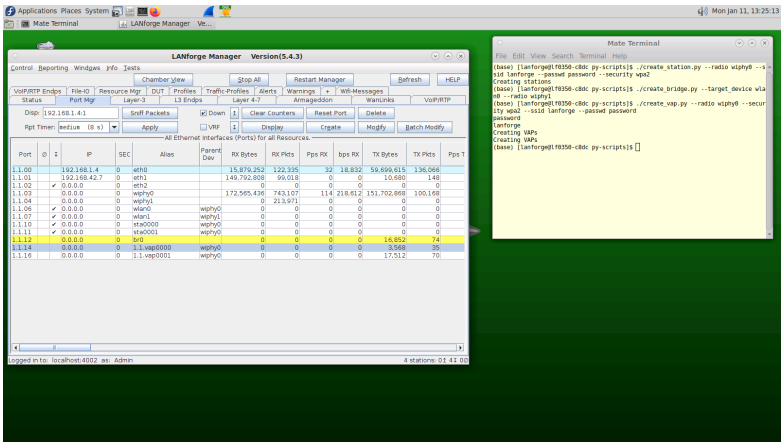
C. create_bridge requires the following arguments:

A. bridge_name - Name of the bridge to create

B. target_device - which device the bridge is going to connect

D. It is not valid to add stations to a bridge, they don't work like you would expect. Bridges can have: eth ports, redirects (rdd) ports, vaps, and qvlan ports. Ports in a bridge cannot have IP addresses.

9. VAP Creation



A. You can create a VAP from your terminal with the create_vap.py script.

B. Type

```
./create_vap.py --radio wiphy0 --security wpa2 --ssid lanforge --passwd password
```

into your terminal, remembering to change the ssid, passwd, and security fields to match your network credentials.

C. create_vap supports the following flags:

A. num_vaps - A user defined number of VAPs to create off of your antenna

B. upstream_port - if your ethernet cable to your router is not eth1, define it using this flag. Eth1 is the default for this flag.

C. debug - call this flag if you want detailed diagnostic information

10. MAC-VLAN Creation

A. In the py-scripts folder, there is a script named create_macvlan.py, which creates a mac-vlan based on the *macvlan_parent* , *num_ports* , *first_macvlan_ip*, *netmask*, and *gateway* input

11. Monitor Creation: Under Construction

12. Layer-3 Cross Connect

A. In your py-scripts, there is a create_l3 script which allows you to create stations in your terminal.

B. Type

```
./create_l3.py --radio wiphy0 --num_stations 0
```

into your terminal, remembering to change the ssid, passwd, and security fields to match your network credentials.

C. This will create a layer3 cross connect between each station you already built on your lanforge device and your wiphy0 radio. You might need to wait 30 seconds for them to stop being phantom ports. If you do not specify num_stations 0 it would have created two stations by default off your specified radio. You can change which port the cross connect will be connected to with the --upstream_port option.

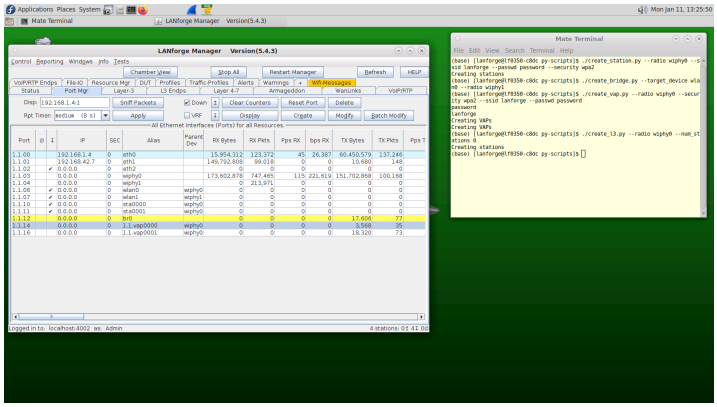
D. You can specify the following flags as well:

A. num_stations - Specify a different number of stations to create off of your antenna.

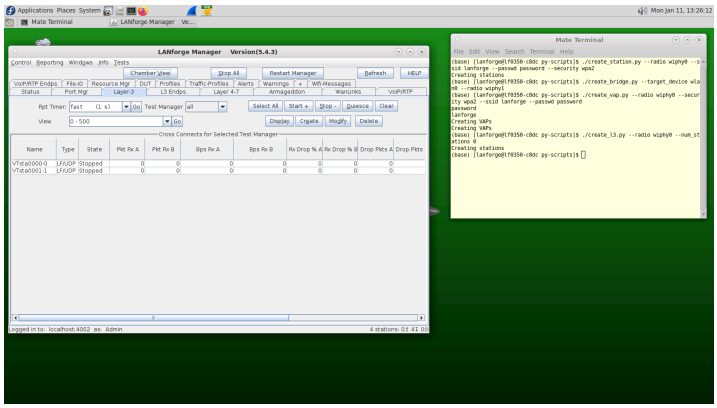
B. upstream_port - if your ethernet cable to your router is not eth1, define it using this flag. Eth1 is the default for this flag.

C. debug - call this flag if you want to get error messages in case anything goes wrong.

E. Your port manager will look similar to this



F. Your Layer 3 connections will look similar to this



13. Layer-4 Cross Connect

A. In your py-scripts, there is a create_l4 script which allows you to create stations in your terminal.

B. Type

```
./create_l4.py --radio wiphy0 --ssid lanforge --passwd password --security wpa2
```

into your terminal

C. This will automatically create 2 stations on your Lanforge device off of your wiphy0 radio and also create a cross connect from each station to your eht1 port. You can change which port the cross connect will be connected to with the --upstream_port option.

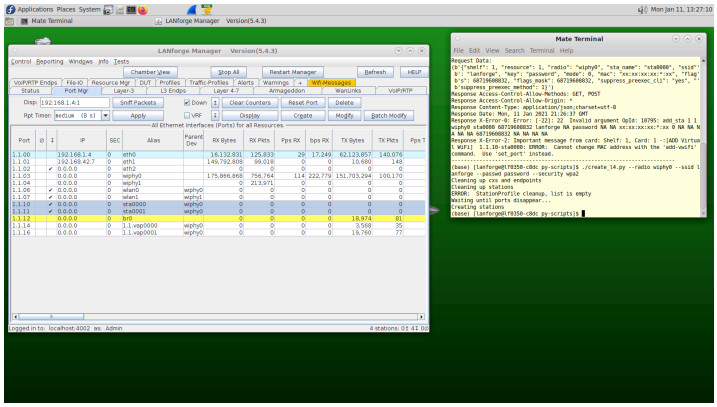
D. You can specify the following flags as well:

A. num_stations - Specify a different number of stations to create off of your antenna.

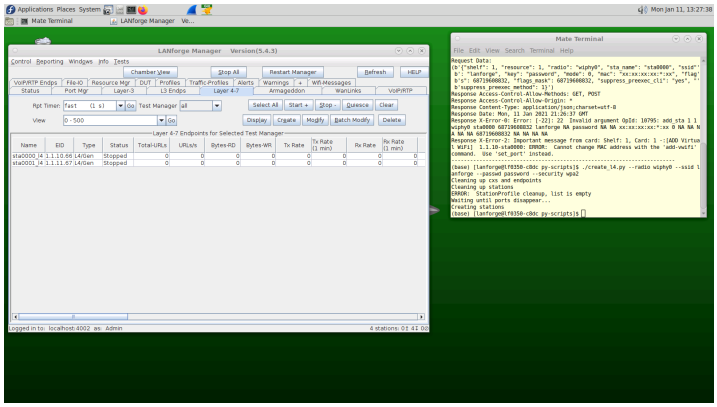
B. upstream_port - if your Ethernet cable to your router is not eth1, define it using this flag. Eth1 is the default for this flag.

C. debug - call this flag if you want detailed diagnostic information

E. Your port manager will look similar to this



F. Your Layer 4 connections will look similar to this



G. Your Netsmith display will look similar to this. The connections on this picture have been oriented for legibility.

14. Monitor and record an IPV4 variable time test. The purpose of this test is to detect whether your router is able to keep a steady signal when being barraged by multiple users.

    A. In the first part of this tutorial, you are going to connect various numbers of stations, record them for 1 and 10 minutes, and then save as a CSV, excel, or pickle format. A station represents a device which is connected to a network, LANforge creates representations of stations which create real traffic on your network and then records statistics on that traffic. This module determines whether your device is able to sustain a heavy load of traffic for an user defined period of time. By recording the traffic it is then possible to go back and detect where any problems occurred which allows a network manager to fix problems which could be facing your network.

    B. Navigate to the py-scripts folder and type the following command into your command line
    `./test_ipv4_variable_time.py --radio wiphy0 --security wpa2 --ssid lanforge --password password --output_format csv`
    Replace the security, ssid, and password variables with the settings for the network you are testing.This will create 2 wiphy stations by default, connect them to the network you are testing, and report the results to a CSV file.You will find a file with a timestamp within the last 5 minutes in the report_data folder in your home directory. If you are running this script from another machine using the --mgr function, you will need to define the report_file variable.

    C. test_ipv4_variable_time accepts the following flags:

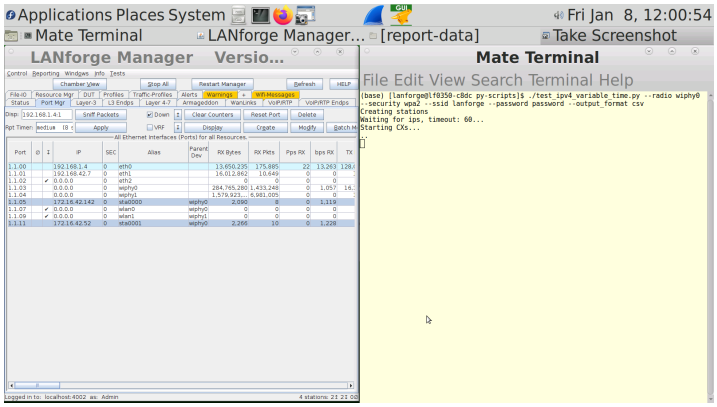
        A. output_format - The format you want to save your results to

        B. col_names - Which columns should be saved in the output file

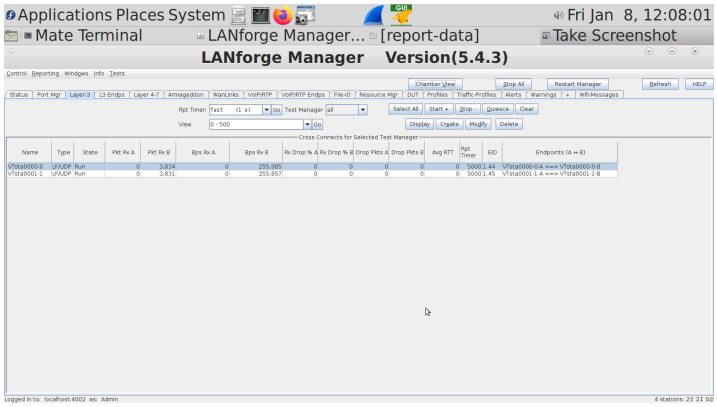
        C. test_duration - how long you want the test to last.

        D. report_file - where you want the results to be stored

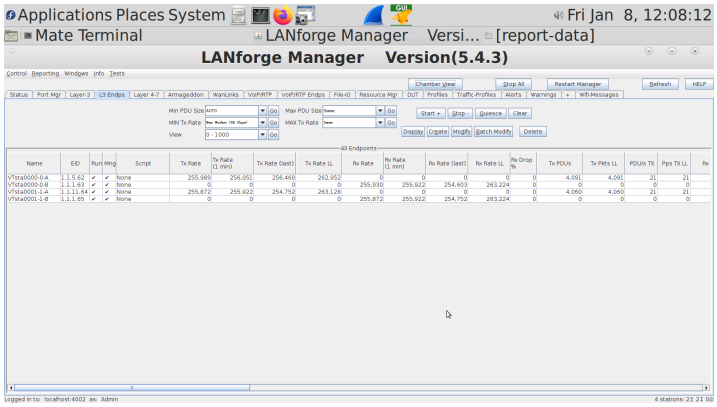
    D. Your port manager will look similar to this



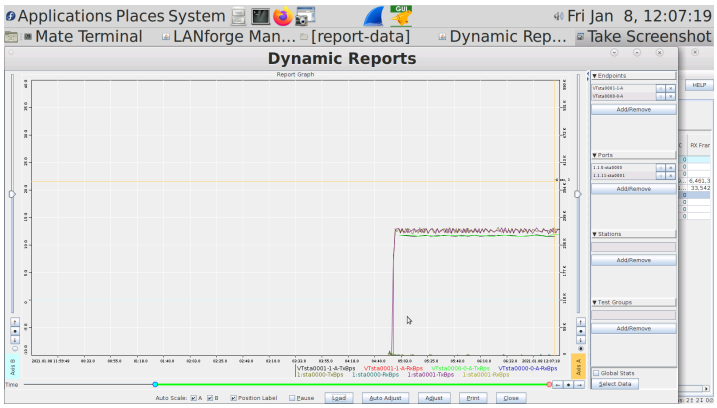
    E. Your Layer 3 connections will look similar to this



    F. Your Layer 3 endpoints will look similar to this



G. You can view a dynamic report of the connectiosn creating traffic. It will look similar to this



Querying the LANforge JSON API using Python

Goal: Use Python scripts to query the LANforge Client JSON API. (See Querying the LANforge GUI for JSON Data) The provided Python scripts allow you the same API scope as the Perl scripts.

LANforge now provides Python scripts that query the REST API that the LANforge Client now exposes by default. This chapter steps through using each of the scripts. At the end we show an example of how to write a Python script. Scripts encourage Python 3. Requires LANforge 5.4.1 or later.

Client Settings

On your LANforge Server, the scripts directory is located at `/home/lanforge/scripts`. Under that directory, the `py-json` directory contains Python scripts.

Script	Purpose
__init__.py	Defines the py-json module
show_ports.py	Simplest example of querying all ports. This is typical of querying any of the APIs you browse from / or /help
create_sta.py	Script creates a virtual station
LANforge/__init__.py	Defines the py-json/LANforge module
LANforge/LFRequest.py	Use the LFRequest module to help make GET and POST requests to the LANforge Client
LANforge/LFUtils.py	Use the LFUtils module to help process JSON results

Getting Started

First, start the LANforge Client (LANforge GUI) and connect it to your LANforge Server. If you want to start the client in headless mode, open a terminal, and from the LANforgeGUI_5.4.1 directory, start the script with the -daemon argument:

```
$ ./lfclient -daemon -s localhost
```

Querying Ports

Running the script

In the `/home/lanforge/scripts/py-json` directory:

```
$ python3 ./show_ports.py
{ '1.1.eth0': { '_links': '/port/1/1/0',
                'alias': 'eth0',
                'phantom': False,
```

```
                'port': '1.1.00'}}
{ '1.1.eth1': { '_links': '/port/1/1/1',
                'alias': 'eth1',
                'phantom': False,
                'port': '1.1.01'}}
{ '1.1.sta00500': { '_links': '/port/1/1/9',
                    'alias': 'sta00500',
                    'phantom': False,
                    'port': '1.1.09'}}
{ '1.1.sta00501': { '_links': '/port/1/1/10',
                    'alias': 'sta00501',
                    'phantom': False,
                    'port': '1.1.10'}}
{ '1.1.sta00502': { '_links': '/port/1/1/11',
                    'alias': 'sta00502',
                    'phantom': False,
                    'port': '1.1.11'}}
```

Looking inside the script

This script is a way of pretty-printing the results of `GET http://localhost:8080/port/1/1/list`

```
lf_r = LFRequest.LFRequest("http://localhost:8080/port/1/1/list")
json_response = lf_r.getAsJson()
j_printer = pprint.PrettyPrinter(indent=2)
for record in json_response['interfaces']:
     j_printer.pprint(record)
```

Other variations of this you can try are:

/port/list
: This is an abbreviation

/port/1/2/list
: If you have a second LANforge resource

/port/1/2/eth0
: Show a specific port

Example of Creating a Station

Running the script

```
[lanforge@ct524-debbie py-json]$ python3 ./create_sta.py
Example 1: will create stations sta0200,sta0201,sta0202
Ex 1: Checking for station : http://localhost:8080/port/1/1/sta0200
...
Ex 1: Next we create stations...
Ex 1: Next we create station sta0200...
Ex 1: station up sta0200...

Example 2: using port list to find stations
Ex 2: checking for station : sta0220
Ex 2: create station sta0220
Ex 2: set port sta0220

Example 3: bring ports up and down
Ex 3: setting ports up...
Ex 3: setting ports down...
...ports are down
Example 4: Modify stations to mode /a
using add_sta to set sta0200 mode

Example 5: change station encryption from wpa2 to wpa3...
using add_sta to set sta0200 wpa3

Example 7: alter TX power on wiphy0...
```

Looking inside the script

Create a station

Flags are a decimal equivalent of a hexadecimal bitfield you can submit as either 0x(hex) or (dec) a helper page is available at http://localhost:8080/help/add_sta

You can watch console output of the LANforge GUI client when you get errors to this command, and you can also watch the websocket output for a response to this command at ws://localhost:8081. Use `$ wsdump ws://localhost:8081/` to follow those messages.

Modes are listed at http://localhost/LANforgeDocs-5.4.1/lfcli_ug.html or at https://www.candelatech.com/lfcli_ug.html

The MAC address field is a pattern for creation: entirely random mac addresses do not take advantage of address mask matchin in Ath10k hardware, so we developed this pattern to randomize a section of octets:

XX
: keep parent

*****
: randomize

chars [0-9a-f]
: use this digit

If you get errors like "X is invalid hex character", this indicates a previous `rm_vlan` call has not removed your station yet: you cannot rewrite mac addresses with this call, just create **new** stations.

The `staNewDownStaRequest()` creates a station in the Admin-Down state. This is a good way to efficiently create batches of stations because it defers all the PHY layer activity which takes significant time when you do it in a loop.

```
lf_r.addPostData( LFUtils.staNewDownStaRequest(sta_name, resource_id=resource_id, radio=radio, ssid=ssid, passphrase=passphra
lf_r.formPost()
sleep(0.05)
```

Sleeping for 50ms is not sufficient to interact with the station, but is a functional minimum to allow the LANforge to start processing the command; this is a good value to use in a loop that creates stations. Follow with:

```
LFUtils.waitUntilPortsAppear(resource_id, desired_stations)
```

Set station up

The LANforge API separates STA creation and Ethernet port settings. We need to revisit the stations we create and amend flags to add things like DHCP or ip+gateway, admin-{up,down} for sta_name in desired_stations:

```
lf_r = LFRequest.LFRequest(base_url+"/cli-json/set_port")
data = LFUtils.portDhcpUpRequest(resource_id, sta_name)
lf_r.addPostData(data)
lf_r.jsonPost()
sleep(0.05)
```

Set station down

Change station mode

There is not a `set_sta` command. Many LANforge CLI commands do a default **modify** if the entity already exists. This is how we can modify attributes of existing stations. For the mode values, see http://www.candelatech.com/lfcli_ug.php#add_sta

```
for sta_name in desired_stations:
  lf_r = LFRequest.LFRequest(base_url+"/cli-json/add_sta")
  lf_r.addPostData({
      "shelf":1,
      "resource": resource_id,
      "radio": radio,
      "sta_name": sta_name,
      "mode": 1, # 802.11a
  })
  lf_r.jsonPost()
  sleep(0.5)
```

Change station protocol

Flags for `add_sta` and `set_port` are actually 64-bit values. When the values in the command below are read by the `/help/add_sta` page, Javascript cannot deal with integers greater than 32-bits long.

```
lf_r = LFRequest.LFRequest(base_url+"/cli-json/add_sta")
lf_r.addPostData({
   "shelf":1,
   "resource": resource_id,
   "radio": radio,
   "sta_name": sta_name,
   "mode": 0, # mode AUTO

   # sets use-wpa3
   "flags": 1099511627776,

   # sets interest in use-wpa3, wpa2_enable (becomes zero)
   "flags_mask": 1099511628800
})
print("using add_sta to set %s wpa3"%sta_name)
lf_r.jsonPost()
```

Change radio power on radio wiphy0

Virtual stations do not have individual tx power states. You can set the radio transmit power. See http://www.candelatech.com/lfcli_ug.php#set_wifi_radio. The txpower is set through `iwconfig`, so see **man 8 iwconfig**. Power is in dBm, **auto** or **off**.

Not all flags in a JSON request are actually LANforge CLI parameters. The `suppress_preexec_method` parameter is a meta-flag tells the LANforge client to not check that the port exists before issuing the command. You would use this to expedite a script, because a check-port command is synchronous, not intended to be used in a loop.

```
lf_r = LFRequest.LFRequest(base_url+"/cli-json/set_wifi_radio")
lf_r.addPostData({
  "shelf":1,
  "resource":resource_id,
  "radio":radio,
  "mode":NA,
  "txpower": "auto",
  "suppress_preexec_method": "true"
})
lf_r.jsonPost()
```

Seeing Errors

Monitoring for Connection Errors

Use the `wsdump` utility on the LANforge to see the output of system errors and WiFi Events:

```
$ wsdump ws://localhost:8081/
```

The output will be mostly similar to what you see in the WiFi-Messages tab in the GUI:

```
< {"wifi-event":"1.1:  IFNAME=sta0200 <3>CTRL-EVENT-SCAN-STARTED","timestamp":"2019-11-21T16:00:50.095Z"}
< {"wifi-event":"1.1:  IFNAME=sta0200 <3>CTRL-EVENT-NETWORK-NOT-FOUND","timestamp":"2019-11-21T16:00:50.095Z"}
< {"wifi-event":"1.1:  sta0200 (phy #1): scan started","timestamp":"2019-11-21T16:00:50.095Z"}
< {"wifi-event":"1.1:  sta0200 (phy #1): scan finished: 5745, \"\"","timestamp":"2019-11-21T16:00:50.095Z"}
< {"wifi-event":"1.1:  IFNAME=sta0220 <3>CTRL-EVENT-SCAN-STARTED","timestamp":"2019-11-21T16:00:50.095Z"}
< {"wifi-event":"1.1:  IFNAME=sta0220 <3>CTRL-EVENT-NETWORK-NOT-FOUND","timestamp":"2019-11-21T16:00:50.095Z"}
```

The message CTRL-EVENT-NETWORK-NOT-FOUND indicates that the SSID we are attempting to connect to is unavailable.

Interpreting Python HTTP Error Output

It won't be uncommon to find errors similar to this:

```
Url: http://localhost:8080/cli-form/set_port
Error:  <class 'urllib.error.HTTPError'>
Request URL:
'http://localhost:8080/cli-form/set_port'
Request Content-type:
```

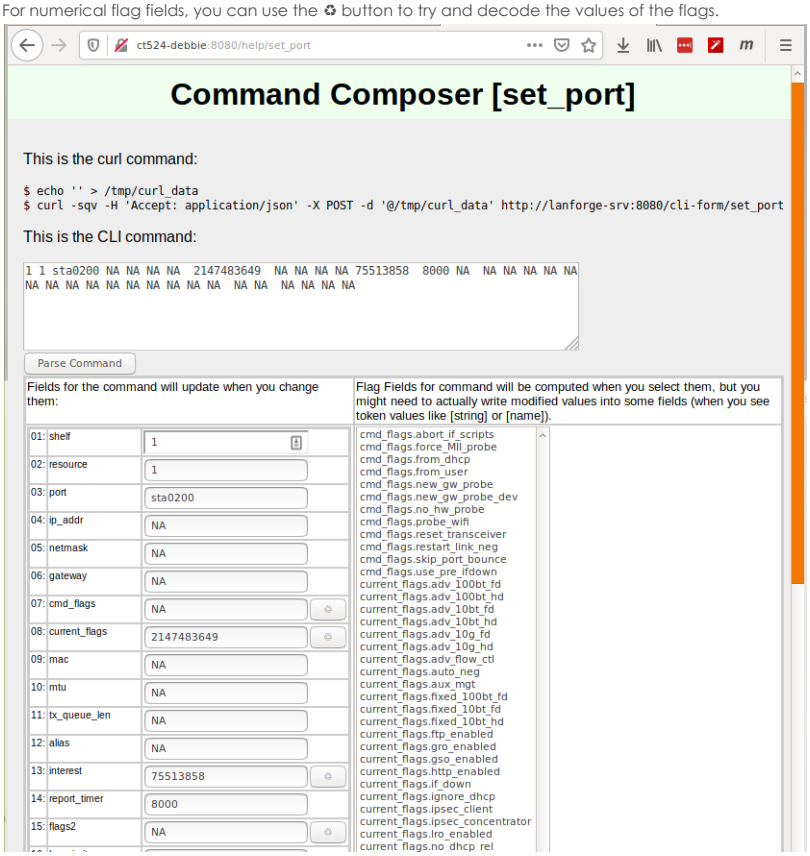
```
'application/x-www-form-urlencoded'
Request Accept:
'application/json'
Request Data:
(b'shelf=1&resource=1&port=sta0200&current_flags=2147483649&interest=75513858&r'
 b'eport_timer=8000')
```

The HTTPError exception is just some kind of 500 error and is often **timing** related. Perl scripts are subject to similar timing issues. When LANforge is busy creating and destroying stations, it is modifying the network stack during each modification...and this takes time.

You can decode this `set_port` request data by pasting the individual values into the `/help/set_port` page provided by your LANforge client: http://localhost:8080/help/set_port.

- shelf 1
- resource 1
- port sta0200
- current_flags 2147483649
- interest 75513858
- report_timer 8000

For numerical flag fields, you can use the ♻ button to try and decode the values of the flags.



Using the Scripts on Your Laptop

You can copy the `py-json` directory to your laptop or workstation. You may also use **git** and clone the LANforge-scripts repository: https://github.com/greearb/lanforge-scripts/.

> i If you make a script on a Windows laptop and copy it back to your LANforge, please run **dos2unix** on the script to change the line-ending characters: `$ dos2unix myscript.py`

Python Module Methods

LFRequest.py

Create a new LFRequest object to help create a request:

```
lf_r = LFRequest.LFRequest(base_url+"/port/1/1/wiphy0")
wiphy0_json = lf_r.getAsJson()
```

Your REST requests are discussed in the Querying LANforge GUI for JSON Data chapter.

formPost(show_error=true)

This method formats post data as `application/x-www-form-urlencoded` data. There should be no significant difference between this and the jsonPost() method.

jsonPost(show_error=true)

This method formats post data as `application/json` data. There should be no significant difference between this and the formPost() method.

get(show_error=true)

Use this method to do a GET request with 'Accept: application/json' headers. You get unformatted results.

getAsJson()

Formats the results of get() into Objects using `json.loads()`

LFUtils.py

def staNewDownStaRequest(sta_name, resource_id=1, radio="wiphy0", flags=ADD_STA_FLAGS_DOWN_WPA2, ssid="", passphrase="", debug_on=False):

For use with add_sta. If you don't want to generate mac addresses via patterns (xx:xx:xx:xx:81:*) you can generate octets using random_hex.pop(0)[2:] and gen_mac(parent_radio_mac, octet) See http://localhost:8080/help/add_sta

def portSetDhcpDownRequest(resource_id, port_name, debug_on=False):
Sets port admin down. See http://localhost:8080/help/set_port

def portDhcpUpRequest(resource_id, port_name, debug_on=False):
Sets port up and to use DHCP. See http://localhost:8080/help/set_port

def portUpRequest(resource_id, port_name, debug_on=False):
Sets port up. See http://localhost:8080/help/set_port

def portDownRequest(resource_id, port_name, debug_on=False):
Sets port down. Does not change the use_dhcp flag See http://localhost:8080/help/set_port

def generateMac(parent_mac, random_octet):
Helps generate a random mac address.

def portNameSeries(prefix="sta", start_id=0, end_id=1, padding_number=10000):
This produces a named series similar to "sta000, sta001, sta002...sta0(end_id)" The padding_number is added to the start and end numbers and the resulting sum has the first digit trimmed, so f(0, 1, 10000) => {0000, 0001}

def generateRandomHex():
Use in conjuction with generateMac()

def portAliasesInList(json_list):
Return reverse map of aliases to port records. Normally, you expect nested records, which is an artifact of some ORM that other customers expect:

```
[
   {
       "1.1.eth0": {
           "alias":"eth0"
       }
   },
   { ... }
]
```

Naturally, this is more difficult to digest. This method returns a more intuitive structure:

```
{
      "eth0" : {
         "1.1.eth0": {
              "alias":"eth0"
           },
       },
       "eth1": {},
       ...
}
```

def findPortEids(resource_id=1, port_names=(), base_url="http://localhost:8080"):
returns PortEID objects matching requested port_names. Use after `set_port`

def waitUntilPortsAdminDown(resource_id=1, port_list=()):
Sleep and query until all ports report admin down. Use after `set_port`

def waitUntilPortsAdminUp(resource_id=1, port_list=()):
Sleep and query until all ports report admin up. Use after `set_port`

def waitUntilPortsDisappear(resource_id=1, port_list=()):
Sleep and query until requested ports have entirely gone away. Use after `rm_vlan`

def waitUntilPortsAppear(resource_id=1, port_list=()):
Sleep and query until requested ports have appeared. Use after `add_sta`

Managing WAN Links Using LANforge JSON API

Goal: Create and modify WAN Links Using LANforge JSON API. This cookbook provides examples in Python. (See Querying the LANforge GUI for JSON Data) The provided Python scripts allow you the same API scope as the Perl scripts.

This chapter steps through using Python scripts to create and manage WAN Links on a LANforge. Scripts require Python 3. Requires LANforge 5.4.1 or later. Examples require CT910, CT521a, or better.

Creating a WAN Link

We will start by creating a WAN Link between ports `eth2` and `eth5` on our ct522. Two Ethernet ports will be involved in this example.

The `create_wanlink.py` Script

This script is located in `/home/lanforge/scripts/py-json` or on the lanforge-scripts github page. You can copy this script to a new name and edit it to fit your enviroment. Remember, these JSON scripts will be querying a LANforge Client (GUI or headless). The URL you will see being queried is going to be `http://localhost:8080/` for these exaples, assuming you are running the LANforge client on the same machine you are running your script.

This script performs the basic tasks you might use to manage WANlink connections:

- Listing existing WANlinks
- Removing a WANlink if it exists.

- Creating WANlink endpoints. You want to create the endpoints before creating the connection.
- Joining WANlink endpoints into a WANlink connection (CX)
- Starting the WANlink
- Modifying the WANlink. You can set endpoint tx rates and lossiness parameters while the endpoints are running.
- Stopping the WANlink
- Listing WANlink endpoint stats

Script Sections Explained

1. **Listing Wanlinks**
   Notice that when we get a listing response, we are looking for items in the response that are dictionarys with a `_links` key/value pair. There are other key/values used for diagnostics, such as *uri, handler, warnings and errors*.

```
base_url = "http://localhost:8080"
json_post = ""
json_response = ""
num_wanlinks = -1
# see if there are old wanlinks to remove
lf_r = LFRequest.LFRequest(base_url+"/wl/list")
try:
   json_response = lf_r.getAsJson()

   # For debugging, this makes json response more legible:
   LFUtils.debug_printer.pprint(json_response)
   for key,value in json_response.items():
      if (isinstance(value, dict) and "_links" in value):
         num_wanlinks = 1
except urllib.error.HTTPError as error:
   num_wanlinks = 0;
```

   JSON output
   If there are no wanlinks, you will only see a warnings block telling you there are connections found that don't apply as WANlinks:

```
{
"warnings" : [
   "HttpWl::selectColumnsFromRow: eid not in table: Cross-Connect cx_id: 17, type: 1, idx: -1",
   "HttpWl::myEvaluateGet: EidCx type 1 (LANforge / UDP) unavailable in WL table: 17.1",
   "HttpWl::selectColumnsFromRow: eid not in table: Cross-Connect cx_id: 18, type: 1, idx: -1",
   "HttpWl::myEvaluateGet: EidCx type 1 (LANforge / UDP) unavailable in WL table: 18.1"
],
"handler" : "candela.lanforge.GenericJsonResponder",
"uri" : "wl/:wl_id"
}
```

2. **Removing a WANlink**
   If we found WANlinks, we can remove them by posting the data to the corresponding CLI command URIs: `/cli-json/rm_cx` and `/cli-json/rm_endp`.

   > i Remember the naming convention: Layer-3 and WANlink endpoints end with `-A` and `-B`. A WANlink named *westin500* has endpoints named *westin500-A* and *westin500-B*.

```
if (num_wanlinks > 0):
   lf_r = LFRequest.LFRequest(base_url+"/cli-json/rm_cx")
   lf_r.addPostData({
      'test_mgr': 'all', # could be 'default-tm', too
      'cx_name': 'wl_eg1'
   })
   lf_r.jsonPost()
   sleep(0.05)
```

   The parameters for each command can be found via the help page: *http://localhost:8080/help/rm_cx*. Notice that slight pause between commands: 50ms is a good idea between deletion commands.

```
   lf_r = LFRequest.LFRequest(base_url+"/cli-json/rm_endp")
   lf_r.addPostData({
      'endp_name': 'wl_eg1-A'
   })
   lf_r.jsonPost()
   sleep(0.05)

   lf_r = LFRequest.LFRequest(base_url+"/cli-json/rm_endp")
   lf_r.addPostData({
      'endp_name': 'wl_eg1-B'
   })
   lf_r.jsonPost()
   sleep(0.05)
```

3. **Creating WANLink Endpoints**
   Create the two endpoints first. Each side of a WANlink has its own transmission rate, buffer size and corruption parameters. Each WANlink requires an ethernet port. Side A will be 128,000bps with 75ms latency:

```
lf_r = LFRequest.LFRequest(base_url+"/cli-json/add_wl_endp")
lf_r.addPostData({
   'alias': 'wl_eg1-A',
   'shelf': 1,
   'resource': '1',
   'port': 'eth3',
   'latency': '75',
   'max_rate': '128000',
   'description': 'cookbook-example'
})
lf_r.jsonPost()
sleep(0.05)
```

   Side B will be 256,000bps with 95ms latency:

```
lf_r = LFRequest.LFRequest(base_url+"/cli-json/add_wl_endp")
lf_r.addPostData({
   'alias': 'wl_eg1-B',
   'shelf': 1,
   'resource': '1',
   'port': 'eth5',
   'latency': '95',
   'max_rate': '256000',
   'description': 'cookbook-example'
})
lf_r.jsonPost()
sleep(0.05)
```

4. ### Create the WANlink
   Creating the WANlink is simple, we will add it to the default test manager *default-tm*:

```
lf_r = LFRequest.LFRequest(base_url+"/cli-json/add_cx")
lf_r.addPostData({
   'alias': 'wl_eg1',
   'test_mgr': 'default_tm',
   'tx_endp': 'wl_eg1-A',
   'rx_endp': 'wl_eg1-B',
})
lf_r.jsonPost()
sleep(0.05)
```

5. ### Start the WANLink
   The LANforge server is very asynchronous. Before immediately changing the state on a connection or endpoint, test to see that it exists.

   Polling for the WANlink
   Note how we can request fields by name, in this case *name, state, and _links*.

```
try:
      json_response = lf_r.getAsJson()
      if (json_response is None):
         continue
```

   If there is no response, or we get a 400 error, the wanlink has probably not finished creating. Our response will be **None**. In the reponse below, we're testing for *dict* entries that have the key `_links` in them. If the name value matches, our WANlink has been created:

```
      for key,value in json_response.items():
         if (isinstance(value, dict)):
            if ("_links" in value):
               if (value["name"] == "wl_eg1"):
                  seen = 1
```

   It might be helpful to use these *else* clauses when getting started:

```
               #else:
               #   print(" name was not wl_eg1")
            #else:
            #   print("value lacks _links")
         #else:
         #   print("value not a dict")

   except urllib.error.HTTPError as error:
      print("Error code "+error.code)
      continue
```

   Change the WANLink State
   Starting and stopping connections is done by changing the state:

```
lf_r = LFRequest.LFRequest(base_url+"/cli-json/set_cx_state")
lf_r.addPostData({
   'test_mgr': 'all',
   'cx_name': 'wl_eg1',
   'cx_state': 'RUNNING'
})
lf_r.jsonPost()
```

   Polling the WANlink State
   The connection might take a second to start. You can poll it similar to to how we polled it above:

```
running = 0
while (running < 1):
   sleep(1)
   lf_r = LFRequest.LFRequest(base_url+"/wl/wl_eg1?fields=name,state,_links")
   try:
      json_response = lf_r.getAsJson()
      if (json_response is None):
         continue
      for key,value in json_response.items():
         if (isinstance(value, dict)):
            if ("_links" in value):
               if (value["name"] == "wl_eg1"):
                  if (value["state"].startswith("Run")):
                     running = 1

   except urllib.error.HTTPError as error:
      print("Error code "+error.code)
      continue
```

6. ### Modifying the WANlink
   The frequency fields below are in occurrance per million. Speeds are set in bits per second (bps). Latencies are in milliseconds.

```
lf_r = LFRequest.LFRequest(base_url+"/cli-json/set_wanlink_info")
lf_r.addPostData({
   'name': 'wl_eg1-A',
   'speed': 265333,        # bps
   'latency': 30,          # 30 ms
   'reorder_freq': 3200,   #  3200/1000000
```

```
   'drop_freq': 2000,      #  2000/1000000
   'dup_freq': 1325,       #  1325/1000000
   'jitter_freq': 25125,   # 25125/1000000
})
lf_r.jsonPost()
```

7. **Stopping the WANlink**

Choose your Stop State
Stopping a WANlink is again, changing its state to either STOPPED or QUIESCE. The QUIESCE state stops transmission on both endpoints but does not close the connection so that in-flight packets can arrive. Choose QUIESCE if you want to make your accounting of packets the most accurate.

```
lf_r = LFRequest.LFRequest(base_url+"/cli-json/set_cx_state")
lf_r.addPostData({
   'test_mgr': 'all',
   'cx_name': 'wl_eg1',
   'cx_state': 'STOPPED'
})
lf_r.jsonPost()
```

Poll Untill Stopped
There might be a milliseconds to seconds of delay depending on how your connection is stopped. You might have to wait for slightly longer than QUIESCE-TIME before the connections are closed when using a QUIESCE stop. Polling the state of the connection is relatively simple:

```
running = 1
while (running > 0):
   sleep(1)
   lf_r = LFRequest.LFRequest(base_url+"/wl/wl_eg1?fields=name,state,_links")
   # LFUtils.debug_printer.pprint(json_response)
```

You might want to watch that debug output at first.

```
   try:
      json_response = lf_r.getAsJson()
      if (json_response is None):
         continue
      for key,value in json_response.items():
         if (isinstance(value, dict)):
            if ("_links" in value):
               if (value["name"] == "wl_eg1"):
                  if (value["state"].startswith("Stop")):
                     LFUtils.debug_printer.pprint(json_response)
                     running = 0
   except urllib.error.HTTPError as error:
      print("Error code "+error.code)
      continue
```

JSON Output

8. **Listing WANlink Endpoint Stats**
Each of the endpoints will show the amount of packets transmitted:

```
   lf_r = LFRequest.LFRequest(base_url+"/wl_ep/wl_eg1-A")
   json_response = lf_r.getAsJson()
   LFUtils.debug_printer.pprint(json_response)
```

JSON Output
The key/value pairs are grouped for this example, but attribute order is not normally ordered.


```
{
   "endpoint" : {
      "name" : "wl_eg1-A",
      "buffer"       : 19936,    # bytes
      "corrupt 1"    : 0,     # these corruptions are per-wanpath
      "corrupt 2"    : 0,
      "corrupt 3"    : 0,
      "corrupt 4"    : 0,
      "corrupt 5"    : 0,
      "corrupt 6"    : 0,
      "delay"        : 30000,
      "dropped"      : 0,
      "dropfreq %"   : 0.200000002980232,
      "dup pkts"     : 0,
      "dupfreq %"    : 0.132499992847443,
      "eid"          : "1.1.3.82",
      "elapsed"      : 7,
      "extrabuf"     : 17408,
      "failed-late"  : 0,
      "jitfreq %"    : 2.51250004768372,
      "maxjitter"    : 0,
      "maxlate"      : 606,
      "max rate"     : 265333,
      "ooo pkts"     : 0,
      "qdisc"        : "FIFO",
      "reordfrq %"   : 0.319999992847443,
      "run"          : false,
      "rx bytes"     : 0,
      "rx pkts"      : 0,
      "script"       : "None",      # applicable if 2544 script has be applied
      "serdelay"     : 45648.296875,
      "tx bytes"     : 0,
      "tx drop %"    : 0,
      "tx pkts"      : 0,
      "tx rate"      : 0,
      "tx-failed"    : 0,
      # below is a to-string debug value
      "wps" : "TblJButton, eid: Shelf: 1  Resource: 1  Port: 3  Endpoint: 82  Type: WanLink",
   },
   "uri" : "wl_ep/:wl_ep_id",
   "candela.lanforge.HttpWlEndp" : {
      "duration" : "0"
```

```
    },
    "handler" : "candela.lanforge.HttpWlEndp$JsonResponse"
}
```

Control a Chamber with the `lf_chamber.pl` Script

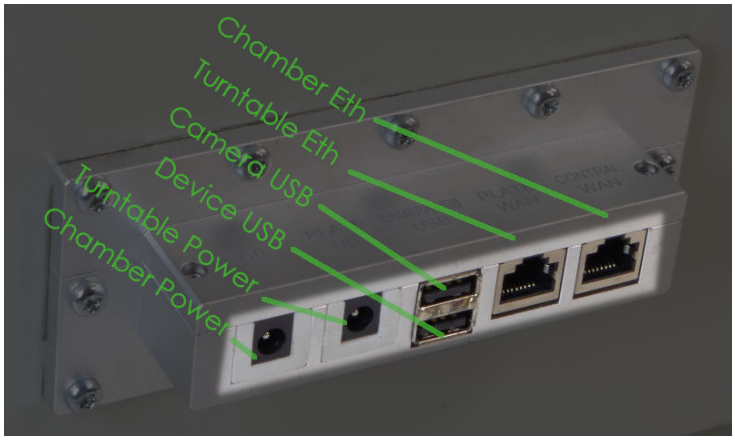
Goal: Monitor and manipulate a CT480a chamber

The CT840a chamber with rotating platform contains an embedded controller that is operated by the modbus protocol. For manual control of the turntable, the LANforge GUI may be used. For automation, you will want to use the `lf_chamber.pl` script.The `lf_chamber.pl` script allows you to monitor the door sensor, table angle, light state and fan state. It also provides control over lights, fans, and table position. This script requires a LANforge server version 5.4.1 or higher to communicate with the chamber. A CT521a or a virtual machine instance is adequate for the task. These instructions apply to the CT840a chamber.

1. ## Configuring the CT840a

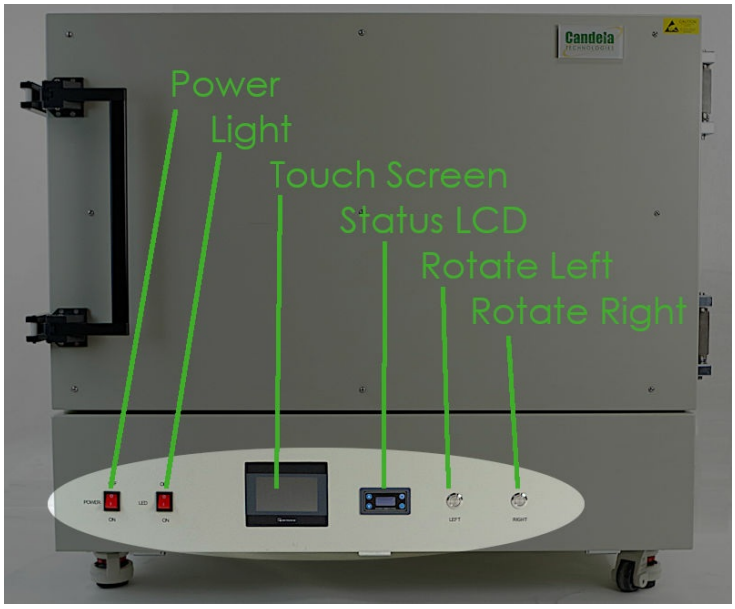
   A. The CT840a requires a network connection. Plug an Ethernet cable into the **Control LAN** port at the bottom rear of the chamber.

   Depending on the date manufacture, these ports might be labeled differently.



   A. The chamber controller and lights are powered via AC cords at the outside bottom rear of the CT840a chamber.
   B. The **Chamber Power** or *DC* port is for 12v or other power required by devices inside the chamber. This runs below the turntable.
   C. The **Turnable Power** or *Plate DC* port is for a 12v or other power required by the DUT on the turntable. This is run up to the top of the turntable.
   D. Accessories or DUTs can be cabled to the **Device USB** port, or *USB* port.
   E. The USB camera has a dedicated USB port, **Camera USB** or *Camera USB* port.
   F. Below the turntable is an Ethernet jack for the DUT to use. That comes out at the **Turnable Eth** or *Plate LAN* port. It should be run up to the top of the turntable with the 12v power cord.
   G. The chamber controller is accessed on the network via the **Chamber Eth** or *Control LAN* port.

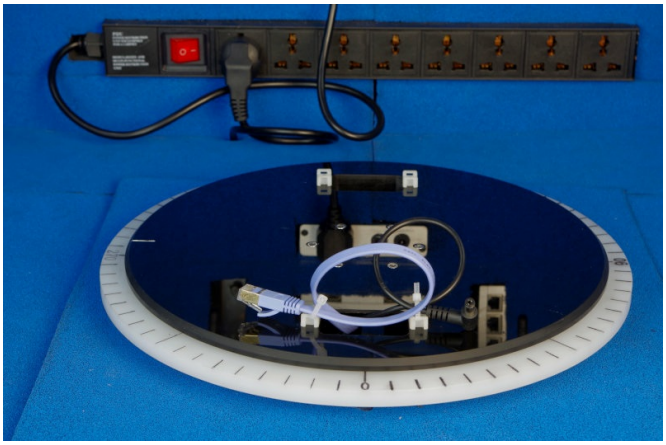
B. Use the front touch screen to set an IP address



C. Make sure you can ping the chamber from your laptop and/or the machine running LANforge server. The LANforge server will communicate over the network to the *Chamber Eth* port.

D. The rear ports are all accessory ports for the chamber.

A. 120v AC cord for internal power strip.



I. This power plug provides power to the chamber modbus controller and the chamber lights.

II. The turn table power cord plugs into the power strip.

B. Pass-through DC barrel connectors. Use these for 12v (or other) power needed by devices in the chamber.

C. SMA connectors. Seal these with terminators when not in use.

D. Ethernet ports

E. USB 3-A and USB C port

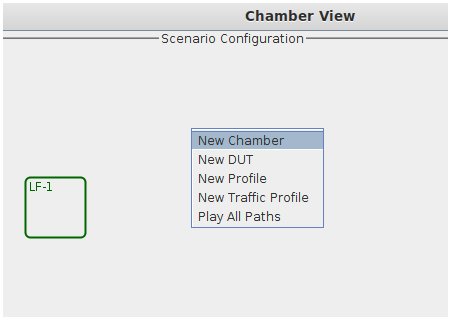
F. Type F Coax port. Seal this with terminators when not in use.

G. Fiber-optic pass-through. Seal this with screw-caps when not in use.

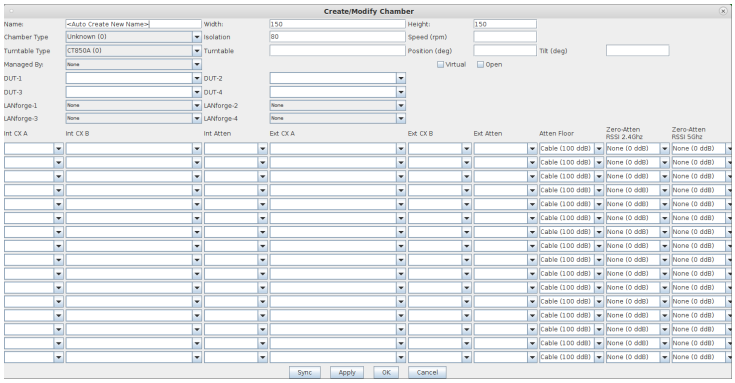
H. HDMI ports

2. ## Configuring the Chamber in LANforge

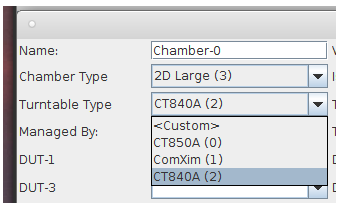
A. In the *Chamber View* window, right-click on the main window and select **New Chamber**



B. You will see the *Create/Modify Chamber* window.



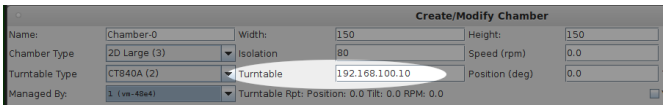
C. Select the chamber and turntable type:



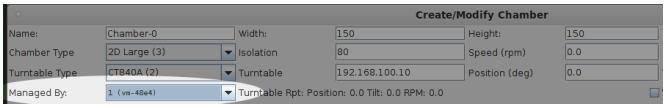
A. For *Chamber Type*, select **2D Large**

B. For *Turntable Type*, select **CT840A**

C. For *Turntable*, put in the IP address of the chamber

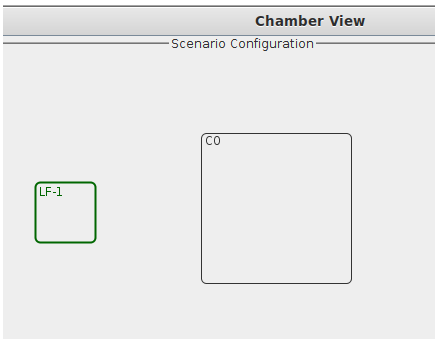


D. Select your LANforge server resource that manages the turntable



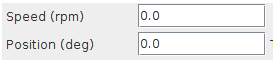
E. Click **OK**

D. You will see a new chamber, **C0** in the *Chamber View* window.



E. In the *Chamber View* window, right-click on the chamber `C0` and select **Modify**

F. Use the *Speed* and *Position* fields to adjust the turntable.



G. Click **Apply** to send the configuration.

3. ## Scripting Chamber Operations

A. Connect to your LANforge system and open a terminal. An ssh connection is adequate.

B. Become root:
$ `sudo -s`

C. The `lf_chamber.pl` script lives in `/home/lanforge`
`cd /home/lanforge`

D. To use the script, you must setup your environment variables located in `/home/lanforge/lanforge.profile`:
`source ./lanforge.profile`

4. Now you may operate the script. Just using the script provides you a summary of options:

```
# ./lf_chamber.pl
Usage:
./lf_chamber.pl --angle 45 --speed 3 --targ 192.168.100.122
./lf_chamber.pl --adjust 5 --targ 192.168.100.122
./lf_chamber.pl --fan 1 --targ 192.168.100.122
./lf_chamber.pl --lights 1 --targ 192.168.100.122
./lf_chamber.pl --status 1 [ --id foo --mgt_pipe /foo/bar ] --targ 192.168.100.122
```

5. The following examples are going to use the example IP `10.0.0.9` for the chamber location on the network.

6. **Chamber Status**

   A. Use the command `./lf_chamber.pl --targ 10.0.0.9 --status 1`

   ```
   Current-Angle: 3598 Door-Open: 0 Table-Moving: 0 Lights: 0 Fan: 0 Jog-Speed: 3 Return-Speed: 3 Absolute-Speed: 3 Jog Angle: 449
   ```

   B. If you see a lot more output, debugging has been enabled. You will see the individual mbpoll commands:

   ```
   Current-Angle: mbpoll -a 1 -r 4139 -t 4 10.0.0.9 -1
       Door-Open: mbpoll -a 1 -r 2094 -t 1 10.0.0.9 -1
       0 Table-Moving: mbpoll -a 1 -r 3046 -t 1 10.0.0.9 -1
       0 Lights: mbpoll -a 1 -r 1283 -t 1 10.0.0.9 -1
       0 Fan: mbpoll -a 1 -r 1284 -t 1 10.0.0.9 -1
       0 Jog-Speed: mbpoll -a 1 -r 4507 -t 4 10.0.0.9 -1
       3 Return-Speed: mbpoll -a 1 -r 4509 -t 4 10.0.0.9 -1
       3 Absolute-Speed: mbpoll -a 1 -r 4511 -t 4 10.0.0.9 -1
       3 Jog Angle: mbpoll -a 1 -r 4513 -t 4 10.0.0.9 -1
   ```

   C. If a script is presently moving the table, you will see an error similar to:

   ```
   Current-Angle: /home/lanforge/local/bin/mbpoll: Connection failed: Operation now in progress.
   COMM-FAIL
   ```

   D. If there is a LANforge service currently engaging the chamber, you might see this error because the lanforge service polls the chamber frequently. If you want to stop the LANforge server, use the command:
   `sudo service lanforge stop`

7. **Controlling the Platform**

   A. Use the command `./lf_chamber.pl --targ 10.0.0.9 --angle 45 --speed 3` to rotate the platform 45 degrees from zero.

   B. You will see the Current-Angle and Jog Angle reported in tenths of degrees, so `450` is 45.0 degrees.#
   `./lf_chamber.pl --targ 10.0.0.9 --status 1`

   ```
   Current-Angle: 450 Door-Open: 0 Table-Moving: 0 Lights: 0 Fan: 0 Jog-Speed: 3 Return-Speed: 3 Absolute-Speed: 3 Jog Angle: 450
   ```

   C. Change argument `--angle 45` to `--adjust 5` to add five more degrees of rotation:

   D. # `./lf_chamber.pl --targ 10.0.0.9 --adjust 5`

   ```
   Adjust 5
   ```

   E. # `./lf_chamber.pl --targ 10.0.0.9 --status 1`

   ```
   Current-Angle: 500 Door-Open: 0 Table-Moving: 0 Lights: 0 Fan: 0 Jog-Speed: 3 Return-Speed: 3 Absolute-Speed: 3 Jog Angle: 50
   ```

   F. To return the plaform to zero rotation, use `--angle 0` argument.

   G. The `--speed` argument modifies the rate of rotation. Speed **1** is very slow; use speed **3** to save time. Speed **6** might be too fast and your DUT might shift unexpectedly.

   H. If you see output that says `mbpoll`, you may ignore those lines.

8. **Controlling the Fans**

   A. Use the command `./lf_chamber.pl --targ 10.0.0.9 --fan 1` to turn fan on.

   ```
   Toggle fan
       /home/lanforge/local/bin/mbpoll -a 1 -r 2074 -t 0 10.0.0.9 -1 0 > /dev/null
       /home/lanforge/local/bin/mbpoll -a 1 -r 2074 -t 0 10.0.0.9 -1 1 > /dev/null
   ```

   # `./lf_chamber.pl --targ 10.0.0.9 --status 1`

   ```
   Current-Angle: 3598 Door-Open: 0 Table-Moving: 0 Lights: 0 Fan: 1 Jog-Speed: 3 Return-Speed: 3 Absolute-Speed: 3 Jog Angle: 44
   ```

   B. Change argument `--fan 1` to `--fan 0` to turn the fan off:

   C. # `./lf_chamber.pl --targ 10.0.0.9 --fan 0`

   ```
   Toggle fan
       /home/lanforge/local/bin/mbpoll -a 1 -r 2074 -t 0 10.0.0.9 -1 0 > /dev/null
       /home/lanforge/local/bin/mbpoll -a 1 -r 2074 -t 0 10.0.0.9 -1 1 > /dev/null
   ```

   D. # **./lf_chamber.pl --targ 10.0.0.9 --status 1**

   ```
   Current-Angle: 3598 Door-Open: 0 Table-Moving: 0 Lights: 0 Fan: 0 Jog-Speed: 3 Return-Speed: 3 Absolute-Speed: 3 Jog Angle: 44
   ```

   E. If you see output that says `mbpoll`, you may ignore those lines.

9. **Chamber Lights**

   A. Use the command `./lf_chamber.pl --targ 10.0.0.9 --lights 1` to turn lights on.

   B. Use the command `./lf_chamber.pl --targ 10.0.0.9 --lights 0` to turn lights off.

   C. The chamber lights are useful when setting up equipment but also for viewing the equipment with the USB camera.

10. **USB Camera**

   A. The USB camera is directly controlled by the connected computer. In the video demonstrations the LANforge system has a USB A-to-USB A cable connected to the chamber to use the camera. The software to use the camera is installed on the LANforge system, it would be one of these: **xawtv, cheese** or **camorama**. There is nothing special about the camera, any laptop should be able to use it. For more information see USB Cable Types

   B. If you want to browse the camera from any machine on the network, the simplest way to do that is to use vncviewer/rdesktop to browse the camera software running on the LANforge desktop.

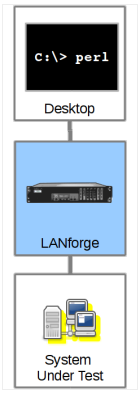
   C. You might notice that the default frame rate of the camera takes a lot of the LANforge CPU time. It should be possible to use `v4l2-ctl` to set the frame rate of the camera.

   D. Please note that recording movies using the camera can be done but they will be very large files. We recommend doing timed frame captures every few seconds to save space.

   E. See also: Video for Linux documentation.

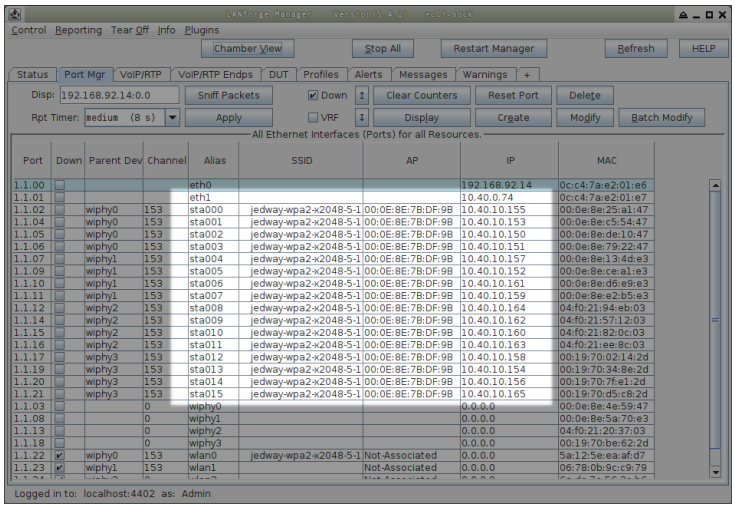
Emulate video streaming traffic with the `l3_video_em.pl` Script

Goal: Emulate video stream traffic patterns using Layer-3 connections.
Using the `l3_video_em.pl` and the `l3_vid_group.pl`, we assemble two test groups of connections, a group of Generic connections, and a group of Layer3 connections, that emulate the bursty buffer filling pattern of traffic that video streaming tends to resemble. Requires LANforge 5.4.2.



1. ## Begin with stations

   A. Using a CT523c, we can create 16 stations, and for this script setup you probably do not want to create more than that. These scripts poll LANforge every 200ms and that loads the server quickly. If you are using a CT521a or CT522b, then consider starting with five or six stations.
   There needs to be a continuously named series of stations.



   For more information see Creating Stations

   B. In this example we will use `eth1` as our **upstream port**. We will be referring to that using the EID format: `1.1.2` For more information see LANforge Entity IDs

2. ## Create Connections Using `l3_vid_group.pl`

A. The script `l3_vid_group.pl` has help examples. You can do four tasks with the script.

```
[lanforge@ct524-genia scripts]$ ./l3_vid_group.pl
Usage: ./l3_vid_group.pl # create a large group of Layer 3 creations that emulate video traffic
 --action -a        { create | destroy | start | stop }
 --buffer_size -b {bytes K|M} # size of emulated RX buffer, default 3MB
 --clear_group -z  # empty test group first
 --cx_name -c       {connection prefix}
 --endp_type -t     {tcp|udp|lf_tcp|lf_udp}
 --first_sta -i     {name}
 --log_cli          {1|filename}   # log cli commands
 --mgr -m           {lanforge server} # default localhost
 --mgr_port -p      {lanforge port} # default 4002
 --num_cx -n        {number} # default 1
 --resource -r      {station resource}
 --speed -s         {bps K|M|G} # maximum speed of tx side, default 1Gbps
 --stream --vid_mode -e {stream resolution name|list} # default yt-sdr-1080p30
                    # list of streams maintained in l3_video_em.pl
 --test_grp -g      {test group name} # all connections placed in this group
                    # default is {cx_name}_tg for the Generic connections
                    # we manage Layer 3 connections in _L3_{cx_name}_tg
 --upstream -u      {port short-EID} # video transmitter port;
                    # use 1.1.eth1 or 1.2.br0 for example
                    # upstream port does not need to be on same resource

Examples:
# create 30 stations emulating 720p HDR 60fps transmitted from resource 2:
 ./l3_vid_group.pl --action create --buffer_size 8M --clear_group --cx_name yt1080p60.1 \
   --endp_type udp --first_sta sta0000 --num_cx 30 \
   --resource 2 --speed 200M --stream yt-hdr-720p60 --test_group yt60fps \
   --upstream 1.2.br0

# start test group:
 ./l3_vid_group.pl -a start -g yt60fps

# stop test group:
 ./l3_vid_group.pl -a stop -g yt60fps

# add 30 more stations on resource 3 to group
 ./l3_vid_group.pl -a create -b 8M -c yt1080p60.3 -t udp -i sta0100 -n 30 -r 3 \
   -s 200M -e yt-hdr-720p60 -g yt60fps -u 1.2.br0

# destroy test group
 ./l3_vid_group.pl -a destroy -g yt60fps
[lanforge@ct524-genia scripts]$
```

A. Create groups of video emulators

B. Start groups

C. Stop groups

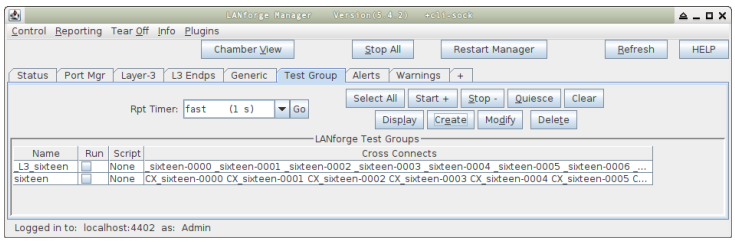
D. Destroy groups

B. We will create a group of 16 connections on our stations. Use the command:

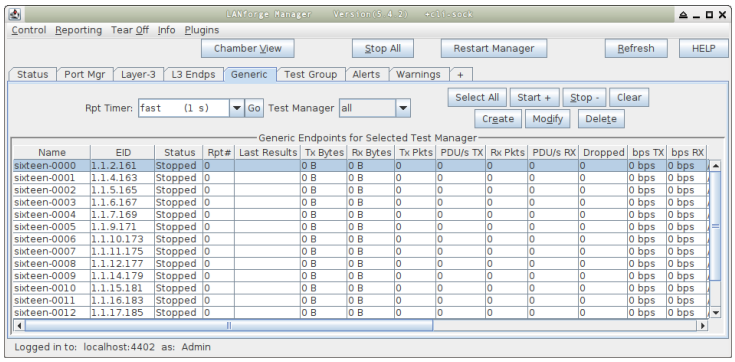
```
./l3_vid_group.pl --action create --endp_type tcp --first_sta sta000
  --num_cx 16 --test_grp sixteen --upstream 1.1.eth1
```

```
[lanforge@ct524-genia scripts]$ ./l3_vid_group.pl --action create --endp_type tcp --first_sta sta000 --num_cx 16 --test_grp sixteen --upstream 1.1.eth1
No cross connects found for test group _L3_sixteen.
No cross connects found for test group sixteen.
Creating test group [sixteen]...Creating test group [_L3_sixteen]...adding L3 CX to _L3_sixteen ................done
Creating Generic connections for video emulation ................done
Adding generic connections to sixteen ................done
[lanforge@ct524-genia scripts]$
```

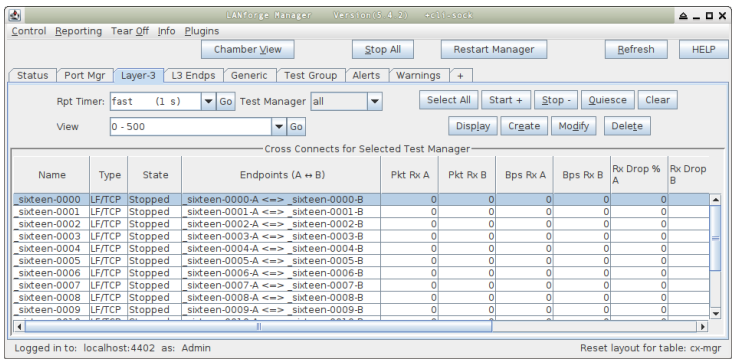
C. You will see two test group created. The group named **_L3_sixteen** contains the Layer-3 tcp connections. The group named **sixteen** contains Generic connections that control the Layer-3 connections.



D. The Generic tab will have 16 connections.



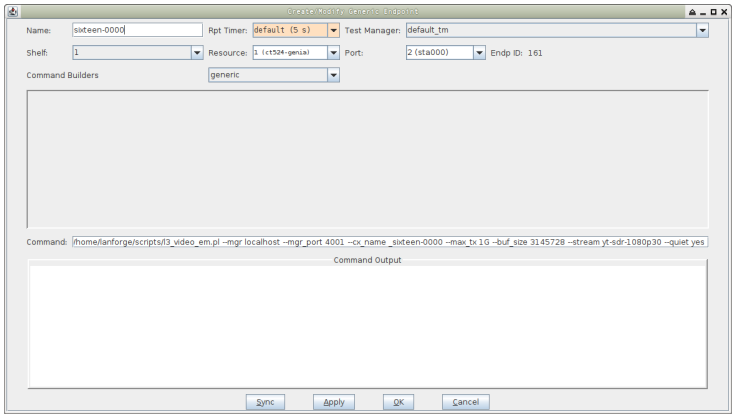
E. The Layer-3 tab will have 16 connections.



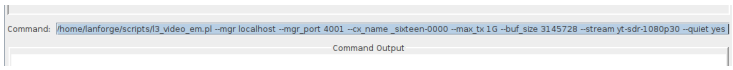
F. When we inspect one of the Generic connections, we can see the command it uses.

```
/home/lanforge/scripts/l3_video_em.pl --mgr localhost --mgr_port 4001
  --cx_name _sixteen-0000 --max_tx 1000000000 --buf_size 3145728 --stream yt-sdr-360p30
  --tx_style bufferfill --quiet yes
```

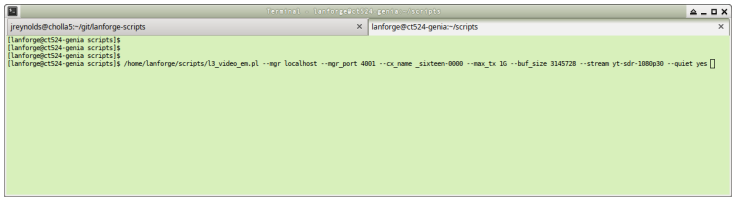
You can paste this command into a shell prompt on your LANforge and use it. We discuss the options in the following section.



G. You can highlight the command in the window and copy it with Ctrl-C

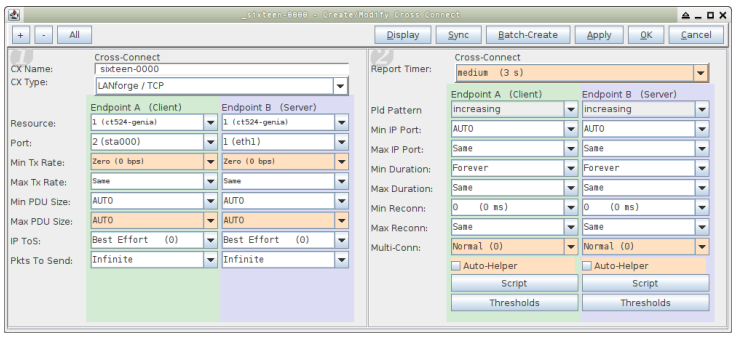


H. You can paste the command into the shell with Ctrl-Shift-V



I. When we inspect the Layer-3 connection, we see these aspects:

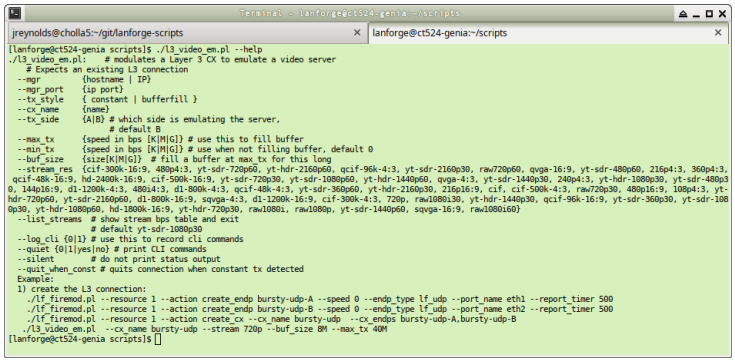
- It is a **TCP** connection. This is optional, you can create UDP connections; the Android YouTube app uses TLS over UDP (QUIC protocol) connections.
- Both endpoints are set at **0 bps** transmit. The Generic script will control the throttle on the B-side of the connection.
- The PDU size is auto. This doesn't have much bearing on TCP, but might have bearing on UDP connections.
- The **Report Timer** is set to three seconds. This value is too long to graph with much detail in the Dynamic Report, but you can shorten it to 500ms if you desire to see more resolution in the Dynamic Report graph. This Report Timer value directly impacts processor load, so use it judiciously.
- Auto-Helper is a new feature intended to reduce CPU load, it has little impact at the moment.
- Multi-con is not desired for this style of connection.



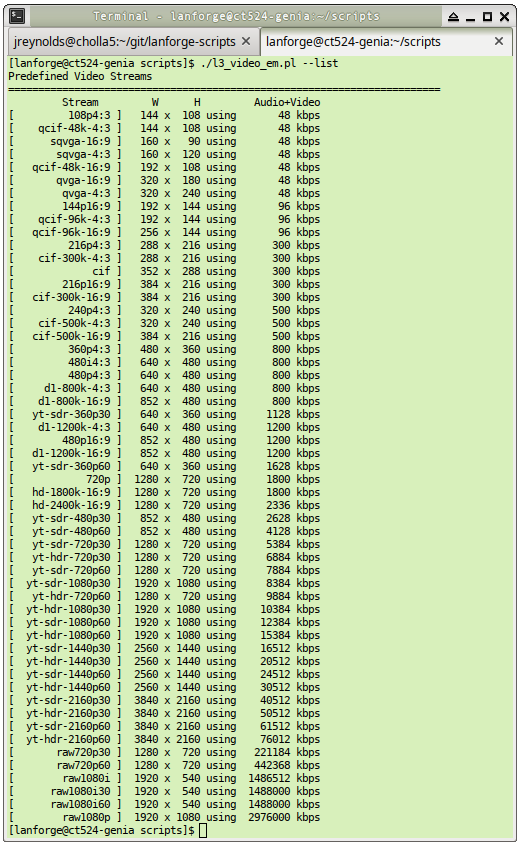
3. ## Exploring `l3_video_em.pl`

A. The options for `l3_video_em.pl` are available with `--help`. The most important options for tuning video streaming emulation are:

- **tx_style**: bufferfill is default and models present video playback
- **max_tx**: this is the starting TX rate. The l3_vid_group script defaults this to 1Gbps, which is unrealistic for common WiFi connections. The script will regularly poll the station side for a TX-RATE value of the station to determine a more realistic upper bound for maximum rate. The more stations that share the same channel, the less realistic this rate becomes. We want to know this to some degree so that we can determine a realistic pause between buffer fills at a given bit rate.
- **buf_size**: observation of packet captures indicate that a video plugin on a browser buffers three to four megabytes of video. Between this number and our max_tx rate, we can calculate when to transmit to fill the video buffer before it empties.
- **stream_res**: This is a list with broadly agreed upon estimates of video bitrates. When people mention frame-rate, that is just part of the bitrate calculation; audio quality, color depth, and resolution are all part of the bitrate value.



B. A table of **stream resolutions** are available when you use the `--list` option. By default, the `l3_vid_group.pl` script uses the **yt-sdr-1080p30** stream size. That name can be decoded like so:
yt: YouTube (but any popular stream, really)
sdr: Standard Dynamic Range color, **hdr**: High Dynamic Range color
1080: frame height
p: progressive, **i:** interlaced
30: 30 frames per second; smaller bitrates might be 29.9, 25, or 24



C. **Running the command**

D. When we run the `l3_video_em.pl` command that we copied and pasted into our terminal above, we'll see regular output ever several seconds. Lets discuss whats going on:

```
Terminal - lanforge@ct524-genia:~/scripts
jreynolds@cholla5:~/git/lanforge-scripts    lanforge@ct524-genia:~/scripts
[lanforge@ct524-genia scripts]$ /home/lanforge/scripts/l3_video_em.pl --mgr localhost --mgr_port 400
1 --cx_name _sixteen-0000 --max_tx 1G --buf_size 3145728 --stream yt-sdr-1080p30 --quiet yes
Filling yt-sdr-1080p30 3072 KB buffer est 0.050331648sec, empties in 3.00164885496183 sec
Random start delay: 3.30490141553221...
Stopping and configuring _sixteen-0000
Starting _sixteen-0000
Likely overfill detected, txsec: 0.0007
## drain_wait_seconds: 1.8043; est fill: 0.1398; actual fill 1.1973; dev: -1.0575
deltas:  Sent 1900544 B/ 2879862464.36994 bps;
Setting max_tx to 360000000
## drain_wait_seconds: 2.2942; est fill: 0.1678; actual fill 0.7074; dev: -0.5397
deltas:  Sent 20971520 B/ 26679081050.06976 bps;, Sent 19333120 B/ 44991395.84851 bps;
Setting max_tx to 300000000
## drain_wait_seconds: 2.3269; est fill: 0.1398; actual fill 0.6747; dev: -0.5349
deltas:  Sent 51576832 B/ 58945207837.85504 bps;, Sent 16515072 B/ 37426482.13639 bps;
Setting max_tx to 360000000
## drain_wait_seconds: 2.2706; est fill: 0.1678; actual fill 0.7311; dev: -0.5633
deltas:  Sent 76742656 B/ 1870580609.91320 bps;, Sent 22151168 B/ 44755255.38064 bps;
Setting max_tx to 300000000
## drain_wait_seconds: 2.3010; est fill: 0.1553; actual fill 0.7006; dev: -0.5453
deltas:  Sent 108199936 B/ 108181984353.88416 bps;, Sent 15728640 B/ 36379923.07554 bps;
Setting max_tx to 324000000
## drain_wait_seconds: 2.2852; est fill: 0.1553; actual fill 0.7164; dev: -0.5611
deltas:  Sent 134021120 B/ 166901817013.20667 bps;, Sent 17367040 B/ 40456583.69786 bps;
Setting max_tx to 324000000
## drain_wait_seconds: 2.3233; est fill: 0.1398; actual fill 0.6784; dev: -0.5386
deltas:  Sent 162791424 B/ 195196318138.62094 bps;, Sent 17498112 B/ 40336066.73510 bps;
Setting max_tx to 360000000
^C
Stopping _sixteen-0000: INT
[lanforge@ct524-genia scripts]$
```

A. **Filling yt-sdr-1080p30 3072KB buffer**: tells us our video bitrate, and our buffer size (3MB)

B. **est 0.0503 sec**: estimate of how long at 1Gbps filling the buffer will take

C. **empties in 3.0016 sec**: playback rate before buffer is fully played.

D. **Random start delay: 3.304sec...**: the script is waiting this long before starting. This is so that we avoid a load spike, false detections of constant transmit, and more realistic transmit pattern.

E. **Likely overfill detected** This warning appears when you transmit longer than your buffer fill takes. Estimates are inaccurate at the start.

F. **drain_wait_seconds** is the computed time between stopping and restarting the next transmission. This is our empty time minus our transmit time.

G. **Actual fill**: describes how long transmitting a full buffer took.

H. **dev**: the difference between estimated fill time and actual fill time.

I. **Setting max_tx to 360000000**: indicates we have detected our RX-Rate for our station was detected, and estimates could be better in range. Estimates are only for an isolated station.

4. ## Starting and Stopping Connections

A. You can use the LANforge GUI *Test Groups* tab or the `l3_vid_group.pl` script to start and stop connections.

B. First, make sure you stations are associated. The scripts will not admin-up your stations.

C. ### Using the LANforge GUI

A. Highlight the Test Group holding the Generic connections. In our example that is the group named **sixteen**

B. Press the **Start** button.

C. The Generic scripts will control the Layer-3 scripts in the **_L3_sixteen** group.

D. Stopping the test group will also stop the Layer-3 connections.

D. ### Using the `l3_vid_group.pl` script

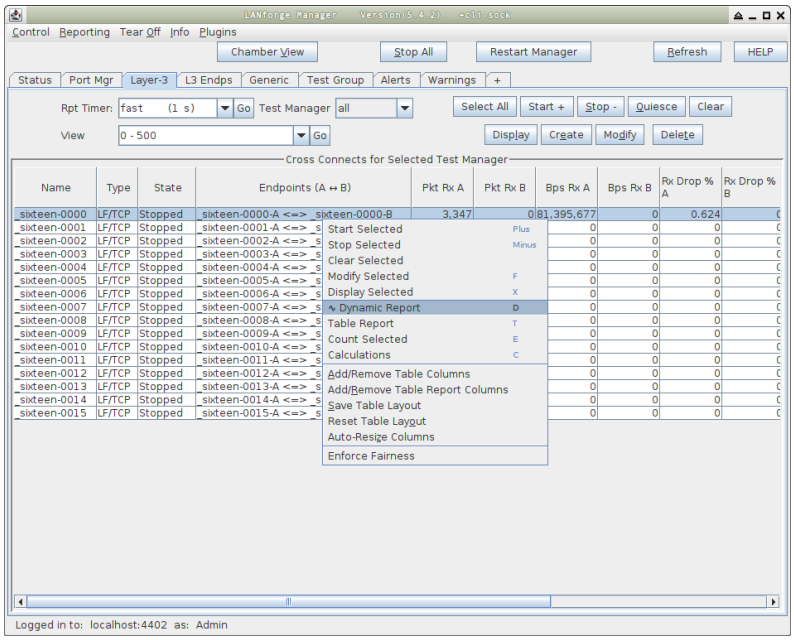
A. Starting the groups uses the `--action start` argument:

```
./l3_vid_group.pl --test_grp sixteen --action start
```

B. Stopping the groups uses the `--action stop` argument:

```
./l3_vid_group.pl --test_grp sixteen --action stop
```

5. ## Observing Performance

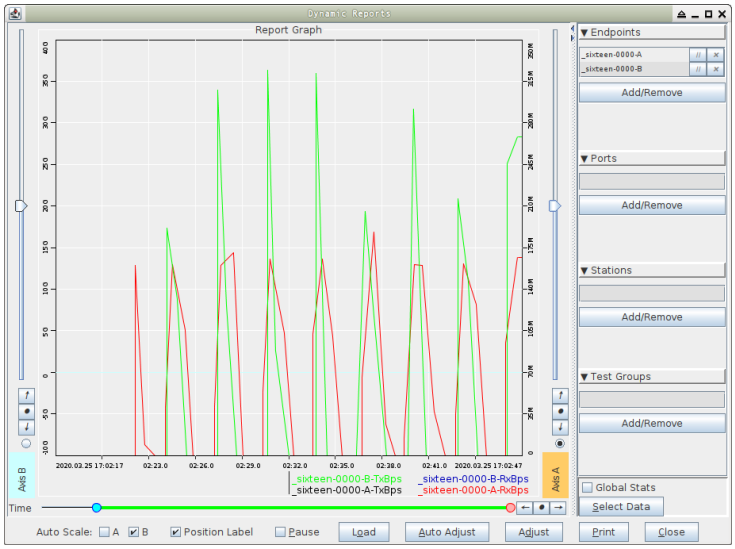


A. You can observe the performance of the Layer-3 connections using the Dynamic Reports window.

B. First, use the **Rpt Timer** combo box to apply a 500ms report timer to the connection. Press **Go** to apply

C. Next, right click the connection and select **Dynamic Report** (or press D)

D. Observe the bursts of transmission.



A. Use the **Adjust** button to adjust your time window:

B. Select **30** for max-time-ago

C. Select **0** for min-time-ago

D. Click **Apply**

Create Python Scripts Utilizing the Realm Library

Goal: Create a python script to create stations and Layer-3 cross connects

Using the `realm.py` library we will write a script that will allow us to automate the creation of stations and Layer-3 cross connects. We will also be able to start and stop traffic over the cross connects using the script. We will be referencing the script, test_ipv4_variable_time.py, as an example throughout this cookbook. Requires LANforge 5.4.2.

1. **Starting the script**

A. Setting up inheritance for our object

   A. In order for our script to be platform independent we will need to `import sys`. Then use

```
if 'py-json' not in sys.path:
    sys.path.append(os.path.join(os.path.abspath('..'), 'py-json'))
```

   B. When creating our object we will need to import the LFCliBase module from the LANforge module using `from LANforge.lfcli_base import LFCliBase`

   C. After importing LFCliBase we can create our Class and inherit from LFCliBase

B. Setting up the main method

   A. The main method will typically follow a pattern:

      I. First, the creation of a list of stations. This can be done in many ways. Example:

```
station_list = LFUtils.port_name_series(prefix_="sta",
    start_id_=0,
    end_id_=4,
    padding_number_=10000)
```

      II. Following the station list, we can initialize our object:

```
ip_var_test = IPV4VariableTime(lfjson_host, lfjson_port,
    number_template="00",
    sta_list=station_list,
    name_prefix="var_time",
    ssid="testNet",
    password="testPass",
    resource=1,
    security="wpa2",
    test_duration="5m",
    side_a_min_rate=256,
    side_b_min_rate=256)
```

      III. After our object has been initialized we can begin the testing process. The preferred order for running our tests is to:

         i. Call `cleanup()` to prevent stations, cross-connects, and endpoints within our list from having creation issues if anything exists with the same name.

         ii. Call the `build()` method in our class to setup the basic versions of the stations, cross-connects, and endpoints.

         iii. Call the `start()` method that will start the test itself, as well as any bring up any stations and start traffic on cross-connects that need it.

         iv. Call the `stop()` method to stop the traffic and bring down any stations that are up.

         v. Verify that the tests passed using our inherited passes() method.

         vi. After verifying a pass we can then call our cleanup function again to clean up everything we worked with.

C. Example Main Method

```
def main():
    lfjson_host = "localhost"
    lfjson_port = 8080
    station_list = LFUtils.portNameSeries(prefix_="sta", start_id_=0, end_id_=4, paddi
    ip_var_test = IPV4VariableTime(lfjson_host, lfjson_port, number_template="00", sta
        name_prefix="var_time",
        ssid="testNet",
        password="testPass",
        resource=1,
        security="wpa2", test_duration="5m",
        side_a_min_rate=256, side_b_min_rate=256),
    ip_var_test.cleanup(station_list)
    ip_var_test.build()
    if not ip_var_test.passes():
        print(ip_var_test.get_fail_message())
        exit(1)
    ip_var_test.start(False, False)
    ip_var_test.stop()
    if not ip_var_test.passes():
        print(ip_var_test.get_fail_message())
        exit(1)
    time.sleep(30)
    ip_var_test.cleanup(station_list)
    if ip_var_test.passes():
        print("Full test passed, all connections increased rx bytes")
```

2. ## Test Methods Available With Realm

A. Using `lfcli_base._pass()` and `lfcli_base._fail()`

   A. Since our class is inheriting lfcli_base.py, we have access to methods that will help us keep track of passes and fails during our tests. We can access them using `self._pass()` or `self._fail()`. They will take two parameters, a string `message` and an optional boolean `print_pass` and `print_fail` for `_pass()` and `_fail()` respectively. If `print_pass` or `print_fail` are set to True, they will write the message to stdout whenever the functions are called.

   B. `lfcli_base` will add a "PASSED: *message*" or "FAILED: *message*" to a list when the tests pass or fail. This list can be accessed using the methods

```
get_result_list()
get_failed_result_list()
get_fail_message()
get_all_message()
```

B. Using lfcli_base to check test success

   A. `passes()` will return a boolean depending on whether or not there were any fails in the test. If it finds a fail message it will return False, if none are found it will return True.
   `get_result_list()` will return all logged pass/fail messages as a list.
   `get_failed_result_list()` will return a list of only fail messages.
   `get_fail_message()` will return a list of string of fail messages separated by newlines
   `get_message()` will return a list of string of all messages separated by newlines

3. ## Building a Station

A. Build Method

A. We will need to do a number of things to setup our build method.

I. To begin we will set the security type of our stations using `station_profile.use_security()`

II. We will then use `station_profile.set_number_template()` to name our stations

III. After this we can set our command flags and parameters using

```
self.station_profile.set_command_flag("add_sta","create_admin_down",1
self.station_profile.set_command_param("set_port","report_timer",1500
self.station_profile.set_command_flag("set_port","rpt_timer", 1)
```

IV. Once our parameters and flags are set, we can pass a list of stations to **`station_profile.create()`** and **`cx_profile.create()`**. Our build function could look like this:

```
for station in range(len(self.sta_list)):
    temp_sta_list.append(str(self.resource)+"."+self.sta_list[station])
self.station_profile.create(resource=1, radio="wiphy0", sta_names_=self.sta_list, debug=Fals
self.cx_profile.create(endp_type="lf_udp", side_a=temp_sta_list, side_b="1.eth1", sleep_time
self._pass("PASS: Station build finished")
```

> i The naming convention for the sides will look like **`foo-A`** for side_a and **`foo-B`** for side_b. *foo* will be set based on the names in the list of stations given.

B. StationProfile

A. The preferred method for creating a station_profile is to use the factory method **`new_station_profile()`** found in realm

I. We will need to assign some variables for the creation of our stations before we can call **`create()`**.

i. `self.station_profile.use_security(`*`security_type`*`, `*`ssid`*`, `*`passwd`*`)` is the preferred method to use when setting the security type, ssid, and password variables
Example:
```
self.station_profile.use_security("wpa2", "testNet", "testPass")
```

ii. **`self.station_profile.number_template_`** is the numerical prefix for stations. Using a **`number_template`** of "00" will have stations look like *sta01, sta02...sta10*
Example:
```
self.station_profile.number_template_="00"
```

iii. **`self.station_profile.mode`** determines the wifi mode used by the stations. See here for available modes
Example:
```
self.station_profile.mode=0
```

4. ## Cross Connects

A. Starting and Stopping Traffic

A. In order for us to be able to start traffic, our stations will need to be admined up, associated, and with an IP. We can bring them up using **`station_profile.admin_up()`**. We can then use **`realm.wait_for_ip(resource, sta_list)`** to wait for our stations, as well as eth1, to get an IP address.

B. Once we are sure all of our stations have ip addresses, we can use **`cx_profile.start_cx()`** to start the traffic for our cross-connects. When we decide to stop the traffic we can just as easily use **`cx_profile.stop_cx()`** to stop traffic.

B. L3CXProfile

A. **`self.local_realm.create_new_l3_cx_profile()`** is the preferred method for creating a new Layer 3 CX Profile.

I. We will need to assign some variables for the creation of our stations before we can call **`create()`**.

i. **`self.cx_profile.name_prefix`** will be used to specify the name prefix for the cx. Assigning **`self.cx_profile.name_prefix`** to "test_" would produce cross-connects named test_sta00 with the numbers being dependent on station_profile's **`number_template`**.
Example:
```
self.cx_profile.name_prefix="test_"
```

ii. Set the **`_min_bps`** to the desired amount. **`_max_bps`** can be set but typically defaults to 0 which sets it to the same as the minimum bps.
Example:
```
self.cx_profile.side_a_min_bps=56000
self.cx_profile.side_b_min_bps=56000
```

5. ## Using TTLS

A. TTLS setup requires a few pieces of information to work correctly. **`StationProfile`** has a **`set_wifi_extra()`** method for setting the relevant variables. See here for the available options

B. We will need a key management type (`key_mgmt`), an EAP method (`eap`), an EAP identity string (`identity`), an EAP password string (`passwd`), an 802.11u realm (`realm`), an 802.11u domain (`domain`), and an 802.11u HESSID (`hessid`)

Example:

```
key_mgmt="WPA-EAP"
eap="TTLS"
identity="testuser"
passwd="testpasswd"
realm="localhost.localdomain"
domain="localhost.localdomain"
hessid="00:00:00:00:00:01"
```

We can then use these variables to call the `set_wifi_extra()` method

Example:

```
station_profile.set_wifi_extra(key_mgmt, eap, identity, passwd, realm, domain, hessid)
```

6. ## Cleaning Up

A. Cleanup stations and cross connects

A. We have two options for cleaning up everything once we finish:

I. The preferred method to cleanup is to use the individual cleanup methods found in `StationProfile` and `L3CXProfile`. These are `station_profile.cleanup(resource, desired_station_list)` and `cx_profile.cleanup()`. These methods are preferred because they will only delete stations, cross-connects, and endpoints created during the test while leaving others untouched.This is useful if you are running other scripts in the background.

II. The other method for cleanup is to use `Realm`'s `remove_all_stations(), remove_all_endps(), and remove_all_cxs()` methods. These will remove all stations, cxs, and endpoints that exist. These are good for doing a full cleanup, and it is recommended to use them in the order of cx, endpoint, station to prevent potential issues or missed deletions.

7. ## Debugging Stations

A. Debug information for station creation can be output by setting `_debug_on=True` in `StationProfile.create()`

A. There are a few important debug outputs to pay attention to:

I. This is the debug output that appears when using the add_sta command. This is used frequently in `StationProfile.create()`. This debug output will allow you to troubleshoot any **flags** or other information that is being set when creating your stations. It will output the name at the top and the raw JSON data will follow.

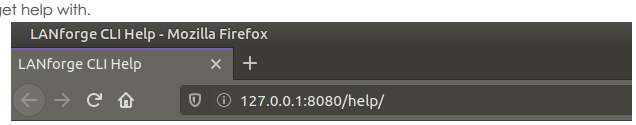
```
- 381 - sta0000- - - - - - - - - - - - - - - - -
{'flags': 132096,
'flags_mask': 68719608832,
'key': 'testPass',
'mac': 'xx:xx:xx:xx:*:xx',
'mode': 0,'
radio': 'wiphy0',
'resource': 1,
'shelf': 1,
'ssid': 'testNet,
'sta_name': 'sta0000'}
```

II. The next bit of debugging output comes from using the set_port command. We are able to see all of the JSON data that is posted, and can use this to check our flags and other info.

```
{'current_flags': 2147483649,
'interest': 8437762,
'port': 'sta0000',
'report_timer': 1500,
'resource': 1,
'shelf': 1}
<LANforge.LFRequest.LFRequest object at 0x7f13dbc56850>
--381 - - - - - - - - - - - - - - - - - -
```

B. There are a few steps we can take to make validating the information we get through debugging easier.

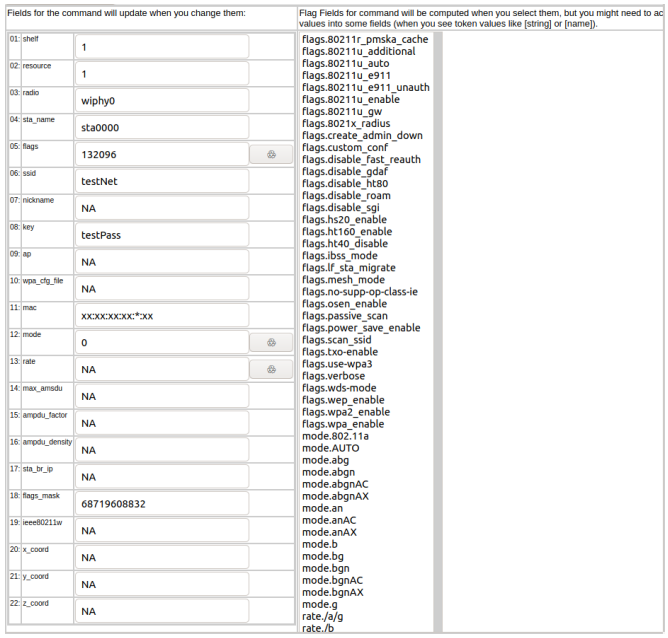
A. We can use the help page available on the address of the machine LANforge is running on. http://127.0.0.1/help/ will take us to a page containing all of the commands we can get help with.



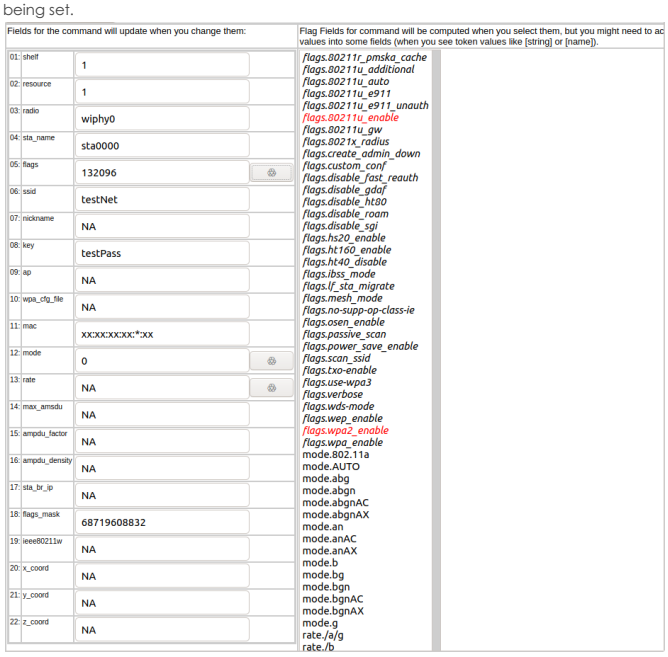
add_arm_endp	CLI Reference for /help/add_arm_endp
add_bgp_peer	CLI Reference for /help/add_bgp_peer
add_bond	CLI Reference for /help/add_bond
add_br	CLI Reference for /help/add_br
add_cd	CLI Reference for /help/add_cd
add_cd_endp	CLI Reference for /help/add_cd_endp
add_cd_vr	CLI Reference for /help/add_cd_vr
add_chamber	CLI Reference for /help/add_chamber
add_chamber_cx	CLI Reference for /help/add_chamber_cx
add_chamber_path	CLI Reference for /help/add_chamber_path
add_channel_group	CLI Reference for /help/add_channel_group
add_cx	CLI Reference for /help/add_cx
add_dut	CLI Reference for /help/add_dut
add_dut_notes	CLI Reference for /help/add_dut_notes
add_endp	CLI Reference for /help/add_endp
add_event	CLI Reference for /help/add_event

B. Using http://127.0.0.1/help/add_sta will bring us to a page specific to the `add_sta` command.

C. Here we can enter all of the data we got from our debugging output into the correct areas.



D. Flag fields have a button next to them that will calculate and highlight relevant flags in the right hand column of the page. This can be useful for checking that the correct flags are being set.



E. After we have done this, we can click the **parse command** button towards the top of the data inputs. We can then enter this command into LANforge's messages tab in the input box.

Create Python Scripts To Test Layer 4 Traffic

Goal: Create a script to test Layer 4 traffic using Realm

Using the `realm.py` library we will write a script that will allow us to automate the creation of stations and Layer 4 cross connects. We will also be able to start and stop traffic over the cross connects using the script. Station and Cross Connect creation is covered in the Realm Scripting Cookbook. Requires LANforge 5.4.2.

---

1. ## Creating The Profile

   A. We will use the factory method `self.local_realm.new_l4_cx_profile()` to create our profile object.

   B. After we have done this we can set a few variables for our traffic:

      A. `l4_cx_profile.requests_per_ten` will set our rate of requests per ten minutes. Setting `requests_per_ten = 600` will set our URL request rate to 1 per second. There is no limit to what can be used as the rate but common rates are:

         - 600 : 1/s
         - 1200 : 2/s
         - 1800 : 3/s
         - 2400 : 4/s

      B. `l4_cx_profile.url` is the URL to be used in the requests. We will also need to specify the direction (dl/ul) and a absolute path for the destination. See syntax here.
         Example:
         ```
         l4_cx_profile.url = "dl http://10.40.0.1 /dev/null"
         ```

      C. Example Layer 4 profile init:
         ```
         class IPV4L4(LFCliBase):
             def __init__(self, host, port, ssid, security, password, url, requests_per
                 target_requests_per_ten=600, number_template="00000", resource=1, num_
                 _debug_on=False,
                 _exit_on_error=False,
                 _exit_on_fail=False):
             super().__init__(host, port, _debug=_debug_on, _halt_on_error=_exit_on_er
             self.host = host
             self.port = port
             self.ssid = ssid
             self.security = security
             self.password = password
             self.url = url
             self.requests_per_ten = requests_per_ten
             self.number_template = number_template
             self.sta_list = station_list
             self.resource = resource
             self.num_tests = num_tests
             self.target_requests_per_ten = target_requests_per_ten

             self.local_realm = realm.Realm(lfclient_host=self.host, lfclient_port=self
             self.cx_profile = self.local_realm.new_l4_cx_profile()
             self.cx_profile.url = self.url
             self.cx_profile.requests_per_ten = self.requests_per_ten

             # Station Profile init
         ```

2. ## Starting Traffic

   A. When running traffic, if you plan to measure the rate of requests, it is recommended to do so in 10 minute increments. An example of this can be seen here: test_ipv4_l4_urls_per_ten.py. To start the traffic we can use the `l4_cx_profile.start_cx()` method. To stop the traffic we can use the `l4_cx_profile.stop_cx()` method.

B. Example start and build method:

```
def build(self):
    # Build stations
    self.station_profile.use_security(self.security, self.ssid, self.password)
    print("Creating stations")
    self.station_profile.create(resource=1, radio="wiphy0", sta_names_=self.sta_list,
    temp_sta_list = []
    for station in range(len(self.sta_list)):
        temp_sta_list.append(str(self.resource) + "." + self.sta_list[station])

    self.l4_profile.create(ports=temp_sta_list, sleep_time=.5, debug_=self.debug, supp
```

```
def start(self, print_pass=False, print_fail=False):
    temp_stas = self.sta_list.copy()
    temp_stas.append("eth1")
    cur_time = datetime.datetime.now()
    interval_time = cur_time + datetime.timedelta(minutes=1)
    passes = 0
    expected_passes = 0
    self.station_profile.admin_up(1)
    self.local_realm.wait_for_ip(self.resource, temp_stas)
    self.l4_profile.start_cx()
    print("Starting test")
    for test in range(self.num_tests):
        expected_passes += 1
        while cur_time < interval_time:
            time.sleep(1)
            cur_time = datetime.datetime.now()

        if self.l4_profile.check_errors(self.debug):
            if self.__check_request_rate():
                passes += 1
            else:
                self._fail("FAIL: Request rate did not exceed 90% target rate", print_
                break
            else:
                self._fail("FAIL: Errors found getting to %s " % self.url, print_fail)
                break
            interval_time = cur_time + datetime.timedelta(minutes=1)
    if passes == expected_passes:
        self._pass("PASS: All tests passes", print_pass)
```

3. ## Examining The Results

A. We can use http://localhost:8080/layer4/list to check our Layer 4 endpoints. Adding a ,,?fields to the end of the URL will allow us to specify what we want to look at. We can separate fields by commas to show more than one at a time.
Example: *http://localhost:8080/layer4/list?fields=name,urls/s,total-urls*

- Using `total-urls` will show us the total requests made.
- Using `urls/s` will show us the average URL rate per second.
- Using `rx rate` and `tx rate` will show us the rates of received and transeferred traffic.

We can also use the url http://localhost:8080/layer4/all to see all of the available fields.

B. When checking our results for Layer 4 tests we might want to check for common URL related errors:

- `acc. denied` will show us the number of times we got an access denied error.
- `bad-url` will show us the number of times a request was made with an invalid URL.
- `nf (4xx)` will count the number of 400 errors recieved when making requests to our URL.

Create Python Scripts To Test Generic Traffic

Goal: Create a script to test Generic traffic using Realm

Using the `realm.py` library we will write a script that will allow us to automate the creation of stations and generic cross connects. We will also be able to start and stop traffic over the cross connects using the script. Station and Cross Connect creation is covered in the Realm Scripting Cookbook. Requires LANforge 5.4.2.

1. ## Creating The Profile

A. We will use the factory method `self.local_realm.new_generic_cx_profile()` to create our profile object.

B. After we have done this we can set a few variables for our traffic:

A. `gen_cx_profile.type` will determine the type of command to execute.
Example: `self.cx_profile.type = "lfping"`

B. `gen_cx_profile.dest` is the destination IP address for the command.
Example: `self.cx_profile.dest = "127.0.0.1"`

C. `gen_cx_profile.interval` sets the interval at which the command is run in seconds.
Example: `self.cx_profile.interval = 1`

D. Example Generic profile init:

```
class GenTest(LFCliBase):
    def __init__(self, host, port, ssid, security, password, sta_list, name_pr
                 number_template="00000", test_duration="5m", type="lfping", c
                 interval=1, radio="wiphy0",
                 _debug_on=False,
                 _exit_on_error=False,
                 _exit_on_fail=False):
        super().__init__(host, port, _debug=_debug_on, _halt_on_error=_exit_on
        self.host = host
        self.port = port
        self.ssid = ssid
        self.radio = radio
        self.upstream = upstream
        self.sta_list = sta_list
        self.security = security
        self.password = password

        self.number_template = number_template
        self.name_prefix = name_prefix
        self.test_duration = test_duration

        self.local_realm = realm.Realm(lfclient_host=self.host, lfclient_port=
        self.cx_profile = self.local_realm.new_generic_cx_profile()
        self.cx_profile.type = type
        self.cx_profile.dest = dest
        self.cx_profile.interval = interval

    # Station Profile init
```

2. ## Starting Traffic

A. To start the traffic we can use the `gen_cx_profile.start_cx()` method. To stop the traffic we can use the `gen_cx_profile.stop_cx()` method.

B. Example start and build method:

```
def build(self):
    self.station_profile.use_security(self.security, self.ssid, self.password)
    self.station_profile.set_number_template(self.number_template)
    print("Creating stations")
    self.station_profile.set_command_flag("add_sta", "create_admin_down", 1)
    self.station_profile.set_command_param("set_port", "report_timer", 1500)
    self.station_profile.set_command_flag("set_port", "rpt_timer", 1)
    self.station_profile.create(radio=self.radio, sta_names_=self.sta_list, debug=self
    self.cx_profile.create(ports=self.station_profile.station_names, sleep_time=.5)
    self._pass("PASS: Station build finished")
```

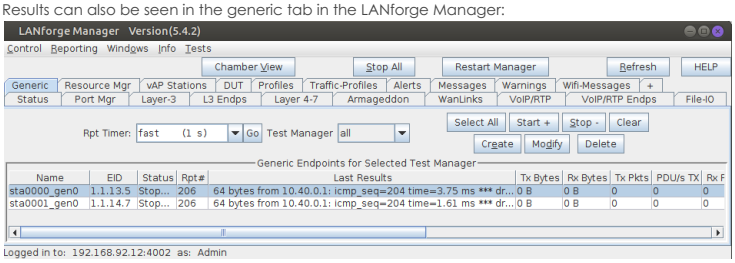
```
def start(self, print_pass=False, print_fail=False):
    self.station_profile.admin_up()
    temp_stas = self.sta_list.copy()
    temp_stas.append(self.upstream)
    if self.local_realm.wait_for_ip(temp_stas):
        self._pass("All stations got IPs", print_pass)
    else:
        self._fail("Stations failed to get IPs", print_fail)
        exit(1)
    cur_time = datetime.datetime.now()
    passes = 0
    expected_passes = 0
    self.cx_profile.start_cx()
    time.sleep(15)
    end_time = self.local_realm.parse_time("30s") + cur_time
    print("Starting Test...")
    while cur_time < end_time:
        cur_time = datetime.datetime.now()
        gen_results = self.json_get("generic/list?fields=name,last+results", debug_=se
        if gen_results['endpoints'] is not None:
            for name in gen_results['endpoints']:
                for k, v in name.items():
                    if v['name'] in self.cx_profile.created_endp and not v['name'].end
                        expected_passes += 1
                        if v['last results'] != "" and "Unreachable" not in v['last re
                            passes += 1
                        else:
                            self._fail("%s Failed to ping %s " % (v['name'], self.cx_p
                            break
        time.sleep(1)
    if passes == expected_passes:
        self._pass("PASS: All tests passed", print_pass)
```

3. ## Examining The Results

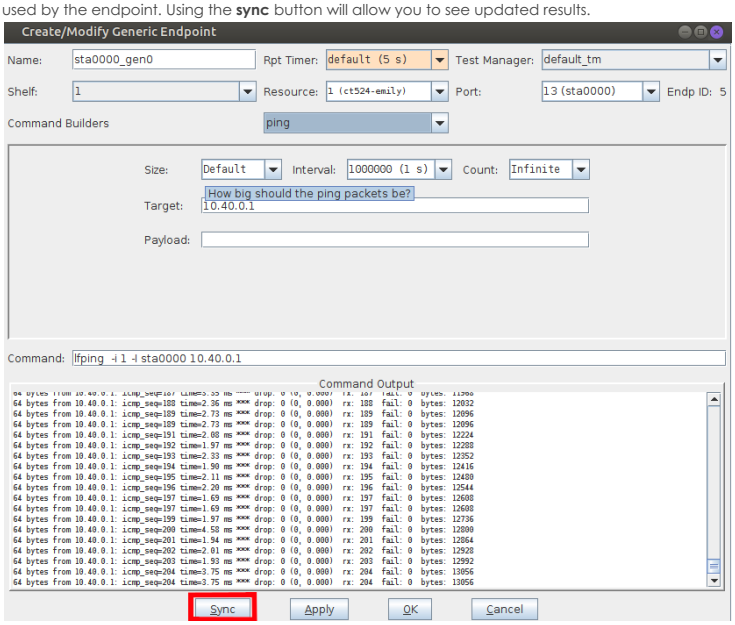
A. For `lfping` we can use the last results of the endpoint to determine if the test was successful. An example of this can be seen in our `start` method. The most common errors for `lfping` will either be a blank last result or `Destination Host Unreachable`. Either of these results indicate a failed ping. Successful pings will look like:

```
64 bytes from 10.40.0.1: icmp_seq=1 time=4.55 ms *** drop: 0 (0, 0.000)  rx: 1  fail:
```

Results can also be seen in the generic tab in the LANforge Manager:



Double-clicking on an endpoint will allow you to see more specific results as well as the command used by the endpoint. Using the **sync** button will allow you to see updated results.



Automate The Creation of VAPs With The Realm Python Library

Goal: Create a python script to create VAPs

Using the `realm.py` library we will write a script that will allow us to automate the creation of VAPs. Requires LANforge 5.4.2

---

1. ## Building a VAP

   A. VAPProfile

      A. The preferred method for creating a `vap_profile` is to use the factory method `new_vap_profile()` found in realm

         I. We will need to set the name of our vap using `vap_profile.vap_name`
         Example:
         ```
         vap_profile.vap_name = "TestNet"
         ```

         II. `vap_profile.use_security(security_type, ssid, passwd)` is the preferred method to use when setting the security type, ssid, and password variables. Available security types are wpa, wpa2, wpa3, wep, and open.
         Example:
         ```
         vap_profile.use_security(type="wpa2", ssid="testNet", passwd="testPass")
         ```

         III. We can change the mode at any time before calling `create()` by modifying the `vap_profile.mode` variable. Changing the mode will allow us to specify the 802.11 wireless standard the VAP uses. See here for available modes.
         Example:
         ```
         vap_profile.mode = 1
         ```

         IV. The channel to be used by the VAP can be set with the channel parameter of the `create()` method.
         Example:
         ```
         vap_profile.create(resource=1, radio="wiphy0", channel=36, up_=True)
         ```

2. ## Bringing VAPs Up/Down

A. `vap_profile.admin_up()` and `vap_profile.admin_down()` can be used to bring the VAP up or down, as necessary.

3. ### Using TTLS

A. TTLS setup requires a few pieces of information to work correctly. `VAPProfile` has a `set_wifi_extra()` method for setting the relevant variables. See here for the available options

B. We will need a key management type (`key_mgmt`), an EAP method (`eap`), an EAP identity string (`identity`), an EAP password string (`passwd`), an 802.11u realm (`realm`), an 802.11u domain (`domain`), and an 802.11u HESSID (`hessid`)
Example:

```
key_mgmt="WPA-EAP"
eap="TTLS"
identity="testuser"
passwd="testpasswd"
realm="localhost.localdomain"
domain="localhost.localdomain"
hessid="00:00:00:00:00:01"
```

We can then use these variables to call the `set_wifi_extra()` method
Example:

```
vap_profile.set_wifi_extra(key_mgmt, eap, identity, passwd, realm, domain, hessid)
```

4. ### Cleaning Up

A. `vap_profile.cleanup()` can be used to remove any VAPs that were created by the profile

Load Scenarios And Control Test Groups With Python

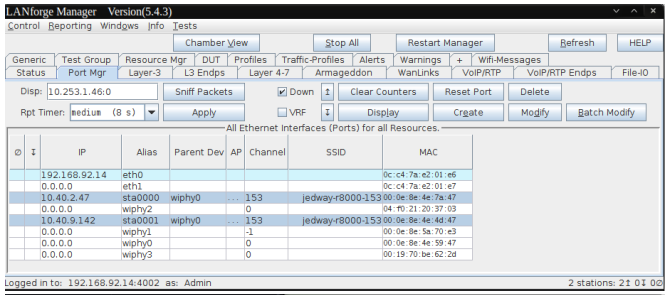
Goal: Using a python script to load scenarios and start, stop, and quiesce test groups
This cookbook will demonstrate how we can use json to load DB scenarios and control test groups using python. We will be referencing the script scenario.py. Requires LANforge 5.4.2.

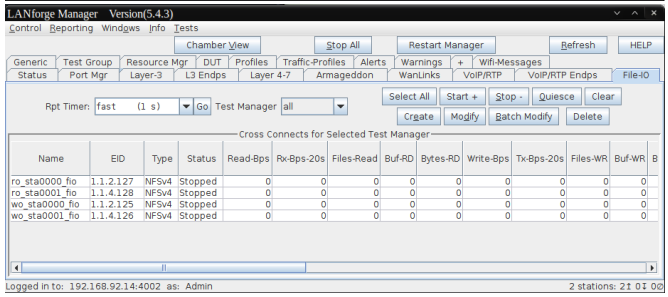
---

1. ### Running The Script

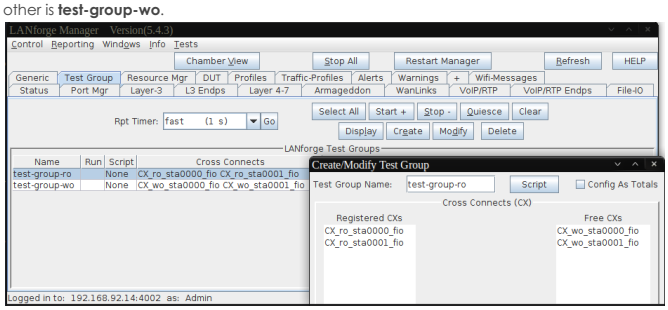
A. Setting up

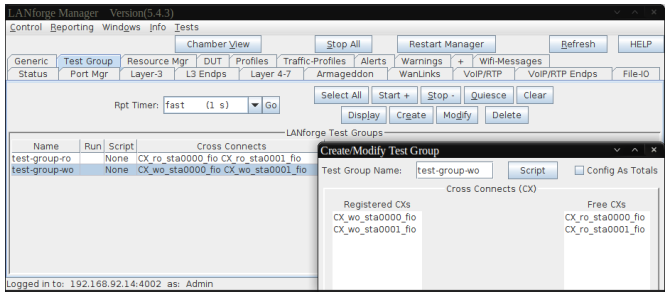
A. For this example we are using a database called fio_test_group. It has two stations that each have a read-only and write-only file-io endpoint attached to them.





Each pair of file-io endpoints are in a group. One group is named **test-group-ro** and the other is **test-group-wo**.





2. ## Script Examples

A. The Command and Available Options

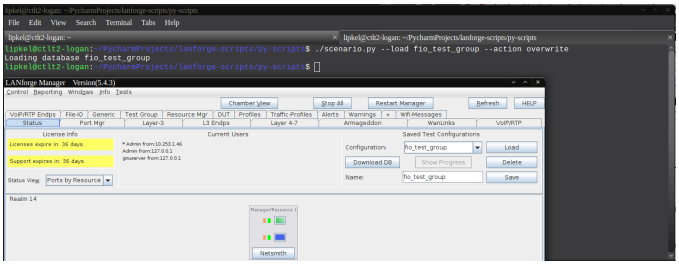
A. The script is located in `lanforge-scripts/py-scripts/`. From that directory we can use `./scenario.py` to run the script.

B. The available options are:

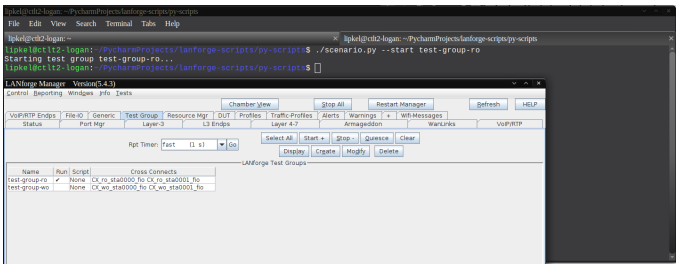
1. `--load db_name` | This will load the database named db_name
2. `--action (overwrite, append)` | Optional argument to be used with `--load`, will specify an action to take when loading the database. The default action is to overwrite. The append option is more difficult to use and its use is discouraged. See here for more info.
3. `--clean_dut` | Optional argument to be used with `--load`, will cleanup DUTs on load. See here for more info.
4. `--clean_chambers` | Optional argument to be used with `--load`, will cleanup Chambers on load. See here for more info.
5. `--start group_name` | This will start the cross-connects in the specified group
6. `--stop group_name` | This will stop the cross-connects in the specified group
7. `--quiesce group_name` | This will quiesce the cross-connects in the specified group

B. Examples of Running the Script

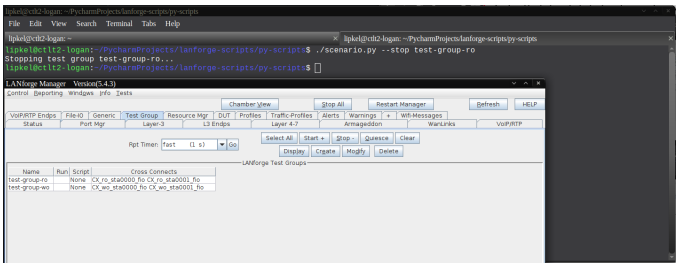
- Loading `fio_test_group` with overwrite



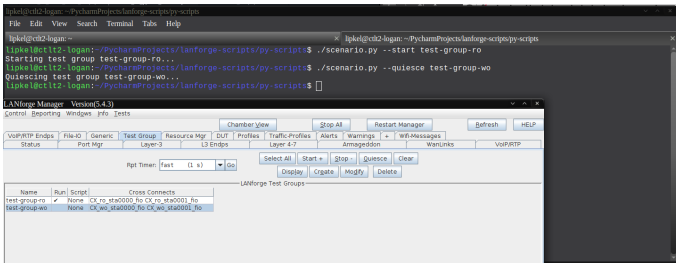
- Starting `test-group-ro`



- Stopping `test-group-ro`



- Quiescing `test-group-wo`



Record the results of a test as CSV from the REALM monitor script

Goal: Record the results of a LANforge test as a CSV file.
Some scripts in the LANforge library have a monitor function built in.
We are going to be using the test_ipv4_variable_time script for this demonstration.
This is useful for running a test and then analyzing the results afterwards.

---

1. Start LANforge GUI. It is recommended to run this script on a fresh LANforge configuration with no stations loaded.
2. Make sure you have lanforge-scripts on your device.
   If lanforge-scripts is already installed on your device, skip this step
   Navigate to py-scripts in the lanforge-scripts folder. If your LANforge device doesn't have this open source software yet you can clone them from Github
   To install lanforge-scripts paste `git clone https://github.com/greearb/lanforge-scripts` into your terminal.
3. Type the following command into your command line
   `./test_ipv4_variable_time.py --radio wiphy0 --security wpa2 --ssid lanforge --password password --output_format csv`
   Replace the security, ssid, and password variables with the settings for the network you are testing.
   This will create 2 wiphy stations by default, connect them to the network you are testing, and report the results to a CSV file.You can change the following fields in the Realm Monitor function:
4. This creates a default file in your report-data folder under your home directory. The name will be in the format with today's timestamp and the name of the test you ran. It's a normal Excel file which you can use however you want..

5. There are multiple commands you can use with this function, here is a list of the flag and what each of them mean:
    A. report_file: Name the full path of the file you want to save results to. Default will save to your report-data folder.
    B. duration_sec: how long you want to run the test
    C. output_format: The output format you want your file in. The following formats are supported:
        A. xlsx DEFAULT
        B. pickle
        HINT: pickle is recommended if you are going to be manipulating data in python since it preserves formatting and can be quickly loaded into a Pandas DataFrame without any manipulation required
        C. csv
        D. json
        E. pdf
        WARNING: PDF is hard to export data from without an Adobe Acrobat license
        F. png
        WARNING: png is going to export an image, do not use this if you are planning on manipulating your data because it does not preserve the numbers recorded
        G. html
        H. hdf
        I. parquet
        J. stata
    D. ssid: REQUIRED Name of the network you are connecting to
    E. password: REQUIRED Password to the network
    F. radio: REQUIRED The radio which you are going to create stations from.
    G. security: Match the security protocol of your router.
    H. test_duration: Default is 60 seconds, write in a any number if you need. You can also use minutes or hours notation in this command, so for 42 minutes write 42m and for 8 hours write 8h.
    I. upstream_port: Most users won't need to use this option, but it tells the program where to connect to the router
    J. created_cx: List of the cross connects you are going to be analyzing. If you are starting with no stations created, you won't need to use this option.

Record the results of a test as an Excel file from the REALM monitor script

Goal: Record the results of a LANforge test as an Excel file.
Some scripts in the LANforge library have a monitor function built in.
We are going to be using the test_ipv4_variable_time script for this demonstration.
This is useful for running a test and then analyzing the results afterwards.

---

1. Start LANforge GUI. It is recommended to run this script on a fresh LANforge configuration with no stations loaded.
2. Make sure you have lanforge-scripts on your device.
   If lanforge-scripts is already installed on your device, skip this step
   Navigate to py-scripts in the lanforge-scripts folder. If your LANforge device doesn't have this open source software yet you can clone them from Github
   To install lanforge-scripts paste `git clone https://github.com/greearb/lanforge-scripts` into your terminal.
3. Type the following command into your command line
   `./test_ipv4_variable_time.py --radio wiphy0 --security wpa2 --ssid lanforge --password password --output_format excel`
   Replace the security, ssid, and password variables with the settings for the network you are testing.
   This will create 2 wiphy stations by default, connect them to the network you are testing, and report the results to an Excel file.
4. This creates a default file in your report-data folder under your home directory. The name will be in the format with today's timestamp and the name of the test you ran. It's a normal Excel file which you can use however you want..
5. There are multiple commands you can use with this function, here is a list of the flag and what each of them mean:
    A. report_file: Name the full path of the file you want to save results to. Default will save to your report-data folder.
    B. duration_sec: how long you want to run the test

C. output_format: The output format you want your file in. The following formats are supported:
    A. xlsx DEFAULT
    B. pickle
       HINT: pickle is recommended if you are going to be manipulating data in python since it preserves formatting and can be quickly loaded into a Pandas DataFrame without any manipulation required
    C. csv
    D. json
    E. pdf
       WARNING: PDF is hard to export data from without an Adobe Acrobat license
    F. png
       WARNING: png is going to export an image, do not use this if you are planning on manipulating your data because it does not preserve the numbers recorded
    G. html
    H. hdf
    I. parquet
    J. stata
D. ssid: REQUIRED Name of the network you are connecting to
E. password: REQUIRED Password to the network
F. radio: REQUIRED The radio which you are going to create stations from.
G. security: Match the security protocol of your router.
H. test_duration: Default is 60 seconds, write in a any number if you need. You can also use minutes or hours notation in this command, so for 42 minutes write 42m and for 8 hours write 8h.
I. upstream_port: Most users won't need to use this option, but it tells the program where to connect to the router
J. created_cx: List of the cross connects you are going to be analyzing. If you are starting with no stations created, you won't need to use this option.

Define and Demonstrate Docstring Usage in Candelatech Python Scripts

Goal: Use PEP 257 standards to properly document python scripts

This cookbook will demonstrate the proper method for documentation in Candelatech created test scripts using PEP 257 guidelines.

Any docstrings occurring after the attribute docstring will be referred to as "additional docstrings". Docstrings in Python are defined as a string literal that is the first statement in a module, function, class or method definition. Such string literals are referred to as "attribute docstrings" and will become the __doc__ attribute of the module, function, class, or method in which they are used.

PEP 257 establishes a standard for docstring usage. In order to keep consistency, triple double quotes should be used for **all** docstrings. Single-line docstrings should be contained entirely on one line. In the example given, a docstring for a function should briefly describe its purpose and specify the return type.

Example taken from PEP 257 page:

```
def function(a, b):
  """Do X and return a list."""
```

Multi-line docstrings should consist of a brief one line summary, followed by a blank line, and finally followed by a more elaborate description. The summary line may either be inline with the opening quotes or on the next line and the whole docstring should be on the same line of indentation as the opening and closing quotations. Closing quotes should exist on their own line, if part of a multi-line docstring, to prevent confusion.

Example taken from PEP 257 page:

```
def complex(real=0.0, imag=0.0):
    """Form a complex number.

    Keyword arguments:
    real -- the real part (default 0.0)
    imag -- the imaginary part (default 0.0)
    """
    if imag == 0.0 and real == 0.0:
      return complex_zero
    ...
```

Implementing Docstring Conventions With Candelatech Script Template

Candelatech scripts will follow PEP 257 specifications for module, function, class or method definitions. To keep things standardized the following example will cover the preferred format for module level docstrings.

```
#!/usr/bin/env python3
"""Module overview/one line description

More detailed summary of module, elaborate on when to use module/
test coverage of full script (Pass/ Fail conditions, columns/ information tested)

External scenario requirements:

Cookbook: http://www.candelatech.com/cookbook.php?vol=cli&book=_______

Copyright 2021 Candela Technologies Inc
License: Free to distribute and modify. LANforge systems must be licensed.
"""
```

Automated scanning of SSID, BSSID, and Signal of available wireless APs

Goal: Create a station and scan for SSID, BSSID, and Signal of available wireless APs

We will learn how to use a script to create a station and scan for available APs. We will then look at the /scanresults/ URI and the info we can get from a scan through JSON. Please refer to sta_scan_test.py as an example script.

---

1. ## Using the Script

   A. Command Line Options

   ```
   --sta_name nameOfStation
   ```

   Specifies the name of the station to be created, if this option is used, the name will default to **sta0000**.

   ```
   --ssid nameOfNetwork
   ```

   Specifies the name of the network to connect to.
   This value must be used, however, the SSID does not have to exist and a fake name can be used.

   ```
   --security {WEP, WPA, WPA2, WPA3, Open}
   ```

   Specifies the security type of the network to connect to.
   This value must be used, however, if a fake SSID is used the type should be open.

   B. Running the script

   A. As an example, we can run the script using:

   ```
   ./sta_scan_test.py --sta_name sta0000 --ssid fake_ssid --security open --radio
   ```

   B. This will produce output that looks like this:

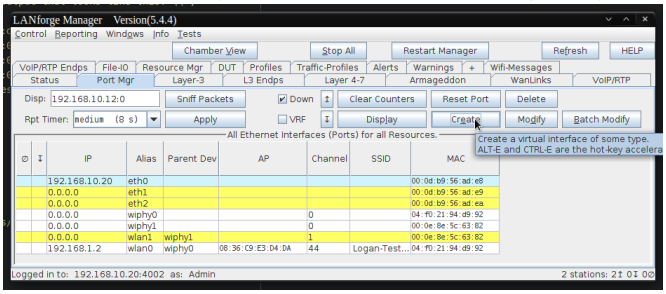
   ```
   BSS                     Signal  SSID
   08:36:c9:e3:d4:da       -32.0   Logan-Test-Net
   10:56:11:0c:04:02       -80.0   :)
   22:56:11:0c:04:02       -79.0   xfinitywifi
   32:56:11:0c:04:02       -80.0   NA
   ```

   This script produces limited output, for more detail we can look at the webpage hosted by LANforge.

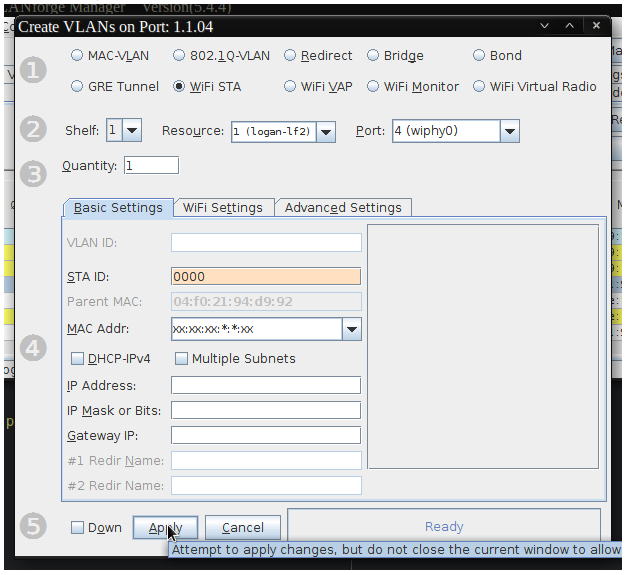
2. ## The /scanresults/ URI

   A. In order to view this page we will need to create a station and start a scan.

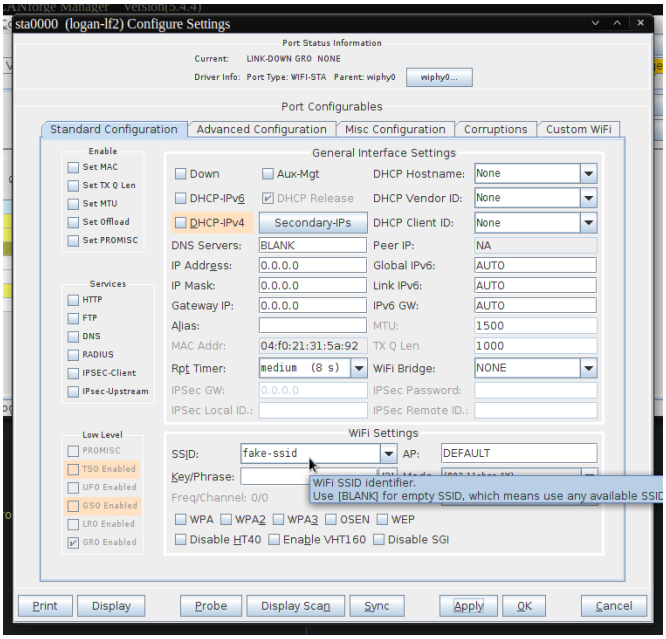
   A. First we will create the station (Make sure to click on a radio in the Port Mgr tab first):



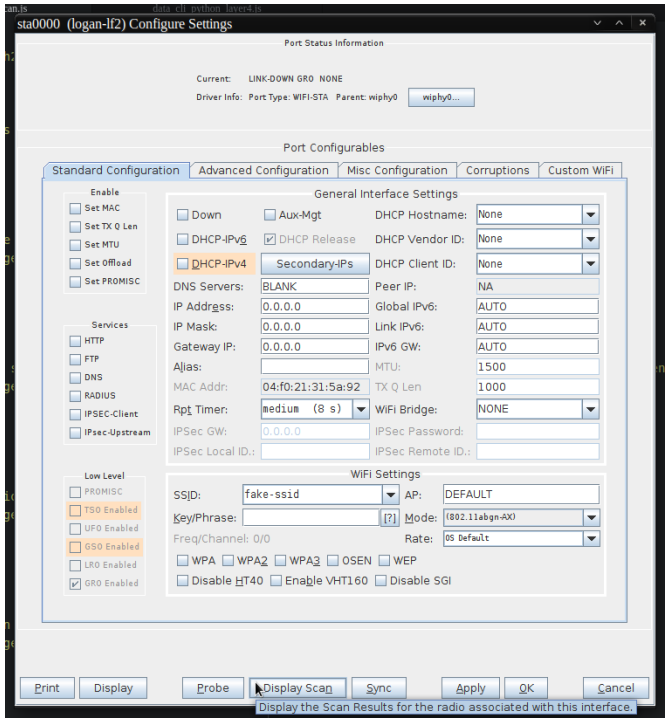
B. Next we will create the station, the default values can be used or a specific number for the station can be given:



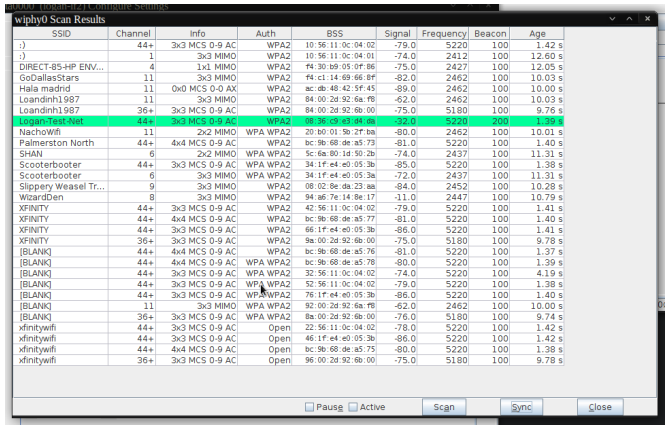
C. After creating the station, we will give the an SSID to connect to. (This doesn't have to be a real AP):



D. Clicking on Display Scan at the bottom of the station settings window will bring us to the Scan window:



E. Finally we'll be able to start the scan and see the results. Clicking on Scan and waiting a few seconds will show all of the APs availble to the station:



SSID	Channel	Info	Auth	BSS	Signal	Frequency	Beacon	Age
:)	44+	3x3 MCS 0-9 AC	WPA2	10:56:11:0c:04:02	-79.0	5220	100	1.42 s
:)	1	3x3 MIMO	WPA2	10:56:11:0c:04:01	-74.0	2412	100	12.60 s
DIRECT-85-HP ENV...	4	1x1 MIMO	WPA2	f4:30:b9:05:0f:86	-75.0	2427	100	12.05 s
GoDallasStars	11	3x3 MIMO	WPA2	f4:c1:14:69:66:8f	-82.0	2462	100	10.03 s
Hala madrid	11	0x0 MCS 0-0 AX	WPA2	ac:db:48:42:5f:45	-89.0	2462	100	10.00 s
Loandinh1987	11	3x3 MIMO	WPA2	84:00:2d:92:6a:f8	-62.0	2462	100	10.03 s
Loandinh1987	36+	3x3 MCS 0-9 AC	WPA2	84:00:2d:92:6b:00	-75.0	5180	100	9.76 s
Logan-Test-Net	44+	3x3 MCS 0-9 AC	WPA2	08:36:c9:e3:d4:da	-32.0	5220	200	1.39 s
NachoWifi	11	2x2 MIMO	WPA WPA2	20:b0:01:5b:2f:ba	-80.0	2462	100	10.01 s
Palmerston North	44+	4x4 MCS 0-9 AC	WPA2	bc:9b:68:de:a5:73	-81.0	5220	100	1.40 s
SHAN	6	2x2 MIMO	WPA WPA2	5c:6a:80:1d:50:2b	-74.0	2437	100	11.31 s
Scooterbooter	44+	3x3 MCS 0-9 AC	WPA WPA2	34:1f:e4:e0:05:3b	-85.0	5220	100	1.38 s
Scooterbooter	6	3x3 MIMO	WPA WPA2	34:1f:e4:e0:05:3a	-72.0	2437	100	11.31 s
Slippery Weasel Tr...	9	3x3 MIMO	WPA2	08:02:8e:da:23:aa	-84.0	2452	100	10.28 s
WizardDen	8	3x3 MIMO	WPA2	94:a6:7e:14:8e:17	-11.0	2447	100	10.79 s
XFINITY	44+	3x3 MCS 0-9 AC	WPA2	42:56:11:0c:04:02	-79.0	5220	100	1.41 s
XFINITY	44+	4x4 MCS 0-9 AC	WPA2	bc:9b:68:de:a5:77	-81.0	5220	100	1.40 s
XFINITY	44+	3x3 MCS 0-9 AC	WPA2	66:1f:e4:e0:05:3b	-86.0	5220	100	1.41 s
XFINITY	36+	3x3 MCS 0-9 AC	WPA2	8a:00:2d:92:6b:00	-75.0	5180	100	9.78 s
[BLANK]	44+	4x4 MCS 0-9 AC	WPA2	bc:9b:68:de:a5:76	-81.0	5220	100	1.37 s
[BLANK]	44+	4x4 MCS 0-9 AC	WPA WPA2	bc:9b:68:de:a5:78	-80.0	5220	100	1.39 s
[BLANK]	44+	3x3 MCS 0-9 AC	WPA WPA2	32:56:11:0c:04:02	-74.0	5220	100	4.19 s
[BLANK]	44+	3x3 MCS 0-9 AC	WPA WPA2	52:56:11:0c:04:02	-79.0	5220	100	1.38 s
[BLANK]	44+	3x3 MCS 0-9 AC	WPA WPA2	76:1f:e4:e0:05:3b	-86.0	5220	100	1.40 s
[BLANK]	11	3x3 MIMO	WPA WPA2	92:00:2d:92:6a:f8	-62.0	2462	100	10.00 s
[BLANK]	36+	3x3 MCS 0-9 AC	WPA WPA2	8a:00:2d:92:6b:00	-76.0	5180	100	9.74 s
xfinitywifi	44+	3x3 MCS 0-9 AC	Open	22:56:11:0c:04:02	-78.0	5220	100	1.42 s
xfinitywifi	44+	3x3 MCS 0-9 AC	Open	46:1f:e4:e0:05:3b	-86.0	5220	100	1.42 s
xfinitywifi	44+	4x4 MCS 0-9 AC	Open	bc:9b:68:de:a5:75	-80.0	5220	100	1.38 s
xfinitywifi	36+	3x3 MCS 0-9 AC	Open	96:00:2d:92:6b:00	-75.0	5180	100	9.78 s

3. ## JSON Response from /scanresults/

A. Another way of viewing the same information is to use the /scanresults/ URI. This URL can be found at your LANforge ip using port 8080. Ex: 192.168.10.20:8080/scanresults. We will also need the shelf number, the resource number, and the station name. The final URL would look like this **192.168.10.20:8080/scanresults/1/1/sta0000**

B. The scan results can be viewed through JSON by using cURL on the same URL as before. The response will look like this:


```
{"handler":"candela.lanforge.HttpStationScan$FixedJsonResponder","uri":"
scanresults/:shelf_id/:resource_id/:port_id","candela.lanforge.HttpStationScan":
{"duration":"1"},"scan-results":[{"1.1.4.08:36:c9:e3:d4:da":{"age":"2238","auth":"WPA2
"beacon":"200","bss":"08:36:c9:e3:d4:da","channel":"44","entity id":"1.1.4",
"frequency":"5220","info":"3x3 MCS 0-9 AC","signal":"-32.0","ssid":"Logan-Test-Net"}}]
```


4. ## Accessing and Printing JSON Response with Python

A. We will use sta_scan_test.py as an example for a `start()` method

A. First, we'll need to send a JSON post using realm. Use this cookbook as reference for getting started with realm. Our JSON will look something like this:

```
data = {
"shelf": 1,
"resource": 1,
"port": self.sta_list
}
```

B. We can then use `json_post` to send the request. We'll need to wait about 15 seconds to give the scan time to happen

```
self.json_post("/cli-json/scan_wifi", data)
time.sleep(15)
```

C. Next, we'll create a variable with the results from the scan using

```
scan_results = self.json_get("scanresults/1/1/%s" % ',.join(self.sta_list))
```

D. Finally, we'll create a loop to iterate through the JSON response and print some nicely formatted output

```
print("{0:<23}".format("BSS"), "{0:<7}".format("Signal"), "{0:<5}".format("SSI
for result in scan_results['scan-results']:
    for name, info in result.items():
        print("%s\t%s\t%s" % (info['bss'], info['signal'], info['ssid']))
```

B. Final Results

A. Our final function will look like this:

```
def start(self):
    self.station_profile.admin_up()
    print(self.sta_list)
    print("Sleeping 15s while waiting for scan")
    data = {
        "shelf": 1,
        "resource": 1,
        "port": self.sta_list
    }
    self.json_post("/cli-json/scan_wifi", data)
    time.sleep(15)
    scan_results = self.json_get("scanresults/1/1/%s" % ',.join(self.sta_list

    print("{0:<23}".format("BSS"), "{0:<7}".format("Signal"), "{0:<5}".format
    for result in scan_results['scan-results']:
        for name, info in result.items():
            print("%s\t%s\t%s" % (info['bss'], info['signal'], info['ssid']))
```

B. Our formatted output should look like this:

```
BSS                     Signal  SSID
00:0e:8e:52:4e:82       -33.0   test-net
08:36:c9:e3:d4:db       -31.0   Logan-Test-Net
08:36:c9:e3:d4:dc       -27.0   Logan-Test-Net
```

Automated Probing of Ports for information

Goal: Probe a port for information on that port.
We will learn how to use a script to probe a port for more information.We will also look at the ouput from the GUI, JSON response, and the script itself. Use the port_probe.py script as a reference.

---

1. ## Using the Script

A. Command Line Options

A. `--port_eid portEID`

Specifies the eid of the port to be probed, if this option is used, the name will default to **1.1.eth0**.

B. Running the script

A. As an example, we can run the script using:

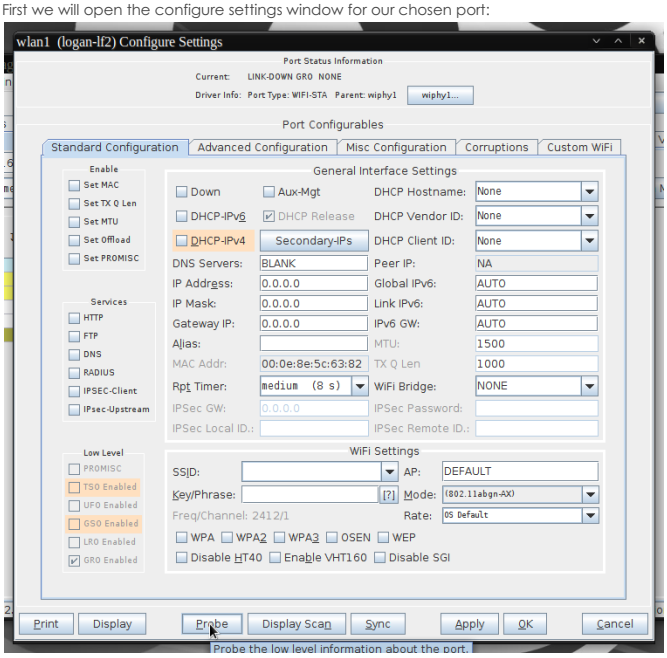
```
./sta_probe_test.py --port_eid 1.1.wlan1
```

This example will probe the existing wlan1 port

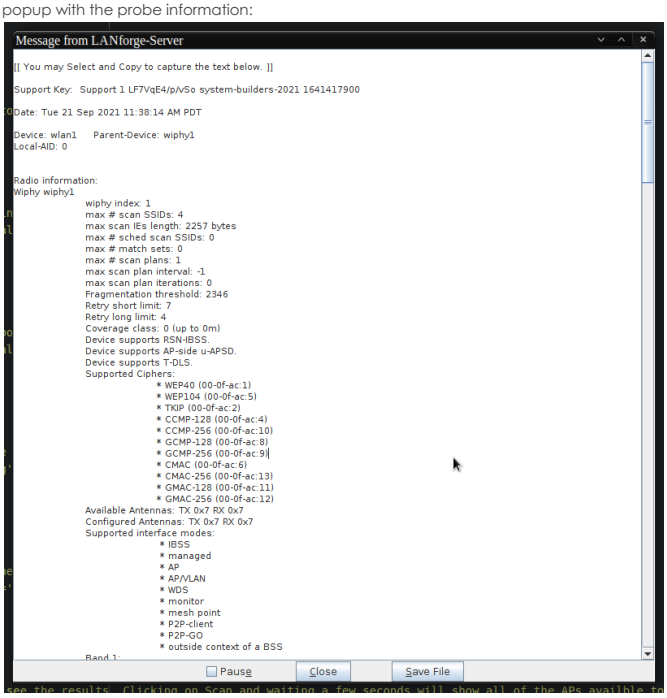
2. ## Probe Results From the GUI

A. In order to view this page we will need to choose a port to use and start probing.

A. First we will open the configure settings window for our chosen port:



B. Next we will click the probe button at the bottom of the window and another window will popup with the probe information:



This information is the formatted version of the probe. The other methods of accessing probe results will be unformatted JSON.

3. ## JSON Response from /probe/

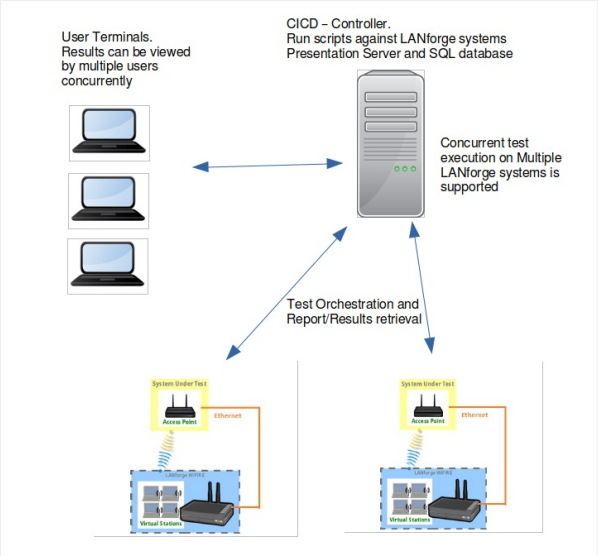
A. Another way of viewing the same information is to access the /probe/ page from LANforge. This can be done by going to the page at your LANforge ip using port 8080. Ex: 192.168.10.20:8080/probe. We will also need the shelf number, the resource number, and the port name.
The final URL would look like this: **192.168.10.20:8080/probe/1/1/wlan1** and the page will look similar to this:

- 1.1.wlan1:
  - entity id: NA
  - probe results: Date: Tue 21 Sep 2021 11:38:14 AM PDT Device: wlan1 Parent-Device: wiphy1 Local-AID: 0 Radio information: Wiphy wiphy1 wiphy index: 1 max # scan SSIDs: 4 max scan IEs length: 2257 bytes max # sched scan SSIDs: 0 max # match sets: 0 max # scan plans: 1 max scan plan interval: -1 max scan plan iterations: 0 Fragmentation threshold: 2346 Retry short limit: 7 Retry long limit: 4 Coverage class: 0 (up to 0m) Device supports RSN-IBSS. Device supports AP-side u-APSD. Device supports T-DLS. Supported Ciphers: * WEP40 (00-0f-ac:1) * WEP104 (00-0f-ac:5) * TKIP (00-0f-ac:2) * CCMP-128 (00-0f-ac:4) * CCMP-256 (00-0f-ac:10) * GCMP-128 (00-0f-ac:8) * GCMP-256 (00-0f-ac:9) * CMAC (00-0f-ac:6) * CMAC-256 (00-0f-ac:13) * GMAC-128 (00-0f-ac:11) * GMAC-256 (00-0f-ac:12) Available Antennas: TX 0x7 RX 0x7 Configured Antennas: TX 0x7 RX 0x7 Supported interface modes: * IBSS * managed * AP * AP/VLAN * WDS * monitor * mesh point * P2P-client * P2P-GO * outside context of a BSS Band 1: Capabilities: 0x11ef RX LDPC HT20/HT40 SM Power Save disabled RX HT20 SGI RX HT40 SGI TX STBC RX STBC 1-stream Max AMSDU length: 3839 bytes DSSS/CCK HT40 Maximum RX AMPDU length 65535 bytes (exponent: 0x003) Minimum RX AMPDU time spacing: 8 usec (0x06) HT TX/RX MCS rate indexes supported: 0-23 Bitrates (non-HT): * 1.0 Mbps * 2.0 Mbps (short preamble supported) * 5.5 Mbps (short preamble supported) * 11.0 Mbps (short preamble supported) * 6.0 Mbps * 9.0 Mbps * 12.0 Mbps * 18.0 Mbps * 24.0 Mbps * 36.0 Mbps * 48.0 Mbps * 54.0 Mbps Frequencies: * 2412 MHz [1] (23.0 dBm) * 2417 MHz [2] (23.0 dBm) * 2422 MHz [3] (23.0 dBm) * 2427 MHz [4] (23.0 dBm) * 2432 MHz [5] (23.0 dBm) * 2437 MHz [6] (23.0 dBm) * 2442 MHz [7] (23.0 dBm) * 2447 MHz [8] (23.0 dBm) * 2452 MHz [9] (23.0 dBm) * 2457 MHz [10] (23.0 dBm) * 2462 MHz [11] (23.0 dBm) * 2467 MHz [12] (disabled) * 2472 MHz [13] (disabled) * 2484 MHz [14] (disabled) Band 2: Capabilities: 0x11ef RX LDPC HT20/HT40 SM Power Save disabled RX HT20 SGI RX HT40 SGI TX STBC RX STBC 1-stream Max AMSDU length: 3839 bytes DSSS/CCK HT40 Maximum RX AMPDU length 65535 bytes (exponent: 0x003) Minimum RX AMPDU time spacing: 8 usec (0x06) HT TX/RX MCS rate indexes supported: 0-23 Bitrates (non-HT): * 6.0 Mbps * 9.0 Mbps * 12.0 Mbps * 18.0 Mbps * 24.0 Mbps * 36.0 Mbps * 48.0 Mbps * 54.0 Mbps Frequencies: * 5180 MHz [36] (20.0 dBm) * 5200 MHz [40] (20.0 dBm) * 5220 MHz [44] (20.0 dBm) * 5240 MHz [48] (20.0 dBm) * 5260 MHz [52] (20.0 dBm) (no IR, radar detection) * 5280 MHz [56] (20.0 dBm) (no IR, radar detection) * 5300 MHz [60] (20.0 dBm) (no IR, radar detection) * 5320 MHz [64] (20.0 dBm) (no IR, radar detection) * 5500 MHz [100] (20.0 dBm) (no IR, radar detection) * 5520 MHz [104] (20.0 dBm) (no IR, radar detection) * 5540 MHz [108] (20.0 dBm) (no IR, radar detection) * 5560 MHz [112] (20.0 dBm) (no IR, radar detection) * 5580 MHz [116] (20.0 dBm) (no IR, radar detection) * 5600 MHz [120] (20.0 dBm) (no IR, radar detection) * 5620 MHz [124] (20.0 dBm) (no IR, radar detection) * 5640 MHz [128] (20.0 dBm) (no IR, radar detection) * 5660 MHz [132] (20.0 dBm) (no IR, radar detection) * 5680 MHz [136] (20.0 dBm) (no IR, radar detection) * 5700 MHz [140] (20.0 dBm) (no IR, radar detection) * 5745 MHz [149] (19.0 dBm) * 5765 MHz [153] (19.0 dBm) * 5785 MHz [157] (19.0 dBm) * 5805 MHz [161] (19.0 dBm) * 5825 MHz [165] (19.0 dBm) * 5845 MHz [169] (disabled) * 5865 MHz [173] (disabled) Supported commands: * new_interface * set_interface * new_key * start_ap * new_station * new_mpath * set_mesh_config * set_bss * authenticate * associate * deauthenticate * disassociate * join_ibss * join_mesh * remain_on_channel * set_tx_bitrate_mask * frame * frame_wait_cancel * set_wiphy_netns * set_channel * set_wds_peer * tdls_mgmt * tdls_oper * probe_client * set_noack_map * register_beacons * start_p2p_device * set_mcast_rate * testmode * connect * disconnect * channel_switch * set_qos_map * set_multicast to unicast Supported TX frame types: * IBSS: 0x00 0x10 0x20 0x30 0x40 0x50 0x60 0x70 0x80 0x90 0xa0 0xb0 0xc0 0xd0 0xe0 0xf0 * managed: 0x00 0x10 0x20 0x30 0x40 0x50 0x60 0x70 0x80 0x90 0xa0 0xb0 0xc0 0xd0 0xe0 0xf0 * mesh point: 0x00 0x10 0x20 0x30 0x40 0x50 0x60 0x70 0x80 0x90 0xa0 0xb0 0xc0 0xd0 0xe0 0xf0 * AP: 0x00 0x10 0x20 0x30 0x40 0x50 0x60 0x70 0x80 0x90 0xa0 0xb0 0xc0 0xd0 0xe0 0xf0 * AP/VLAN: 0x00 0x10 0x20 0x30 0x40 0x50 0x60 0x70 0x80 0x90 0xa0 0xb0 0xc0 0xd0 0xe0 0xf0 * mesh point: 0x00 0x10 0x20 0x30 0x40 0x50 0x60 0x70 0x80 0x90 0xa0 0xb0 0xc0 0xd0 0xe0 0xf0 * P2P-client: 0x00 0x10 0x20 0x30 0x40 0x50 0x60 0x70 0x80 0x90 0xa0 0xb0 0xc0 0xd0 0xe0 0xf0 * P2P-GO: 0x00 0x10 0x20 0x30 0x40 0x50 0x60 0x70 0x80 0x90 0xa0 0xb0 0xc0 0xd0 0xe0 0xf0 Supported RX frame types: * IBSS: 0x40 0xb0 0xc0 0xd0 * managed: 0x40 0xd0 * AP: 0x00 0x20 0xb0 0xc0 0xd0 * AP/VLAN: 0x00 0xb0 0xc0 0xd0 * mesh point: 0xb0 0xc0 0xd0 * P2P-client: 0x40 0xd0 * P2P-GO: 0x00 0x20 0x40 0xb0 0xc0 0xd0 * P2P-device: 0x40 0xd0 software interface modes (can always be added): * AP/VLAN * monitor valid interface combinations: * #{ managed } <= 2048, #{ AP, mesh point } <= 32, #{ P2P-client, P2P-GO } <= 1, #{ IBSS } <= 1, total <= 2048, #channels <= 1, STA/AP BI must match, radar detect widths: { 20 MHz (no HT), 20 MHz, 40 MHz } * #{ WDS } <= 2048, total <= 2048, #channels <= 1, STA/AP BI must match HT Capability overrides: * MCS: ff ff ff ff ff ff ff ff * maximum A-MSDU length * supported channel width * short GI for 40 MHz * max A-MPDU length exponent * min MPDU start spacing Device supports TX status socket option. Device supports HT-IBSS. Device supports SAE with AUTHENTICATE command Device supports low priority scan. Device supports scan flush. Device supports AP scan. Device supports per-vif TX power setting P2P GO supports CT window setting Driver supports full state transitions for AP/GO clients Driver supports a userspace MPM Device supports active monitor (which will ACK incoming frames) Driver/device bandwidth changes during BSS lifetime (AP/GO mode) Device supports ACK timeout estimation. Device supports configuring vdev MAC-addr on create. Supported extended features: * [RIM] RRM * [FILS_STA] STA FILS (Fast Initial Link Setup) * [CQM_RSSI_LIST]: multiple CQM_RSSI_THOLD records * [CONTROL_PORT_OVER_NL80211]: control port over nl80211 * [TXQS]: FQ-CoDel-enabled intermediate TXQs * [AIRTIME_FAIRNESS]: airtime fairness scheduling * [SCAN_RANDOM_SN]: use random sequence numbers in scans * [SCAN_MIN_PREQ_CONTENT]: use probe request with only rate IEs in scans Regulatory information: global country US: DFS-FCC (2402 - 2472 @ 40), (N/A, 30), (N/A) (5170 - 5250 @ 80), (N/A, 23), (N/A), AUTO-BW (5250 - 5330 @ 80), (N/A, 23), (0 ms), DFS, AUTO-BW (5490 - 5730 @ 160), (N/A, 23), (0 ms), DFS (5735 - 5835 @ 80), (N/A, 30), (N/A) (57240 - 71000 @ 2160), (N/A, 40), (N/A) phy#0 country US: DFS-FCC (2402 - 2472 @ 40), (N/A, 30), (N/A) (5170 - 5250 @ 80), (N/A, 23), (N/A), AUTO-BW (5250 - 5330 @ 80), (N/A, 23), (0 ms), DFS, AUTO-BW (5490 - 5730 @ 160), (N/A, 23),

Basic CICD AP Testing with LANforge

Goal: Set up Basic CICD a LANforge system, Regression Automation and Reporting with data from previous runs.

The LANforge CICD framework provides an ability to execute a suite of tests and report results.



---

1. **The following steps are discussed**
   A. **Set Up CICD Controller and Environment**
   B. **Set Up The JSON Configuration Files**
   C. **Test Execution**
   D. **Test Results**

2. **Set Up CICD Controller and Environment**
   A. clone lanforge-scripts from `https://github.com/greearb/lanforge-scripts`
   B. run `/lanforge-scripts/py-scripts/update_dependencies.py` to install python packages for generating output
   C. Install web server:
      The web server is to allow for viewing of results from User Terminals
      The CICD - Controller is not dependent on a web server, results may be viewed locally on CICD - Controller
      A. **LANforge** LANforge installation using kinstall.pl installs a web server on LANforge
         LANforge installation installs an httpd server, LANforge may be used for storing and displaying results.
         For the following example a separate LANforge system (Fedora) was used as the CICD - Controller and httpd web server.
      B. **Fedora** install httpd and configure server `$ sudo dnf install httpd`
      C. **Ubuntu** install apache2 and configure server `$ sudo apt install apache2`
   D. Install mail service for email of links to results
      For the example below Linux mailx program was used
      Installation of mail services is dependent on the environment in which the CICD - Controller is installed.
      The CICD - Controller is not dependent on email services
   E. Install database sqlite3
      A. **Fedora** `$ sudo dnf install sqlite3`
      B. **Ubuntu** `$ sudo apt-get update` `$ sudo apt-get install sqlite3`
   F. Create a html-reports directory. On lanforge /home/lanforge/html-reports
   G. Determine sqlite3 database name and location, sqlite3 db will be created. ./tools/qa_sqlite3.db

3. **Set Up The JSON Configuration Files**
   A. There are three JSON configuration input files described below. For all the JSON configuration files the CAPITALIZED parameters allow for a value to be entered into one location and used in multiple areas of the CICD framework. For example in ssid_indx=1 the SSID_USED is set to asus11ax-5. For the test suite below the SSID_USED may be entered instead of asusu11ax-5, thus if the SSID changes, the SSID will need to be modified in ct_AX88U_dut, the ct_tests.json will remain untouched. This reduces the need to modify the ct_test.json for SSID changes that would affect multiple tests
      A. **--json_rig test_rig.json** this JSON file describes LANforge test rig, Example ct_test_rig.json
         The test_rig.json describes the LANforge system and test parameters for the CICD - Controller

B. **--json_dut ct_AX88U_dut.json** this JSON file describes the AP, Example ct_AX88U_dut.json
the ct_AX88U_dut.json describes the device under test parameters, DUT_SET_NAME:
DUT_NAME ASUSSRT-AX88U for example is used by Chamberview Tests

C. **--json_test ct_tests.json** this JSON file describes the tests, Example ct_tests.json
The tests may use the CAPITALIZED variables or may be entered with the command line arguments as they would be entered on the command line.
The tests are not limited to only python tests

B. **test_rig.json**

```
{
    "test_rig":{
        "Notes":[
            "This JSON file describes LANforge system and test run configuration"
        ]
    },
    "test_rig_parameters":{
        "TEST_BED": "CT-TEST-001",
        "TEST_RIG": "CT-TEST-001",
        "DATABASE_SQLITE": "./tools/qa_sqlite3.db",
        "LF_MGR_IP": "192.168.100.116",
        "LF_MGR_PORT": "8080",
        "LF_MGR_USER": "lanforge",
        "LF_MGR_PASS": "lanforge",
        "UPSTREAM_PORT":"1.1.eth2",
        "TEST_TIMEOUT": 600,
        "EMAIL_LIST_PRODUCTION": "support@candelatech.com",
        "EMAIL_LIST_TEST": "support@candelatech.com",
        "EMAIL_TITLE_TXT": "Lanforge QA Testing",
        "EMAIL_TXT": "Lanforge QA Testing"
    }
}
```

C. **ct_AX88U_dut.json**

```
{
    "ct_AX88U_dut":{
        "Notes":[
            "The device undertest configuration is contained in this file"
        ]
    },
    "test_dut":{
        "DUT_SET_NAME": "DUT_NAME ASUSRT-AX88U",
        "USE_DUT_NAME": "ASUSRT-AX88U",
        "wireless_network_dict":{
            "ssid_idx=0":{"ssid_idx":"0","SSID_USED":"asus11ax-2","SSID_PW_USED":"hello123","BSSID":"3c:7c:3f:55:4d:60","SECUR
            "ssid_idx=1":{"ssid_idx":"1","SSID_USED":"asus11ax-5","SSID_PW_USED":"hello123","BSSID":"3c:7c:3f:55:4d:64","SECUR
        }
    }
}
```

D. **ct_tests.json**

```
{
    "ct_tests_001":{
        "Notes":[
            "This JSON file describes tests to be run by LANforge system"
        ]
    },
    "test_suites":{
        "suite_wc":{
            "create_chamberview_dut_wc":{
                "enabled":"TRUE",
                "load_db":"skip",
                "command":"create_chamberview_dut.py",
                "args":"",
                "args_list":[
                    " --lfmgr LF_MGR_IP --port LF_MGR_PORT --dut_name DUT_NAME",
                    " --ssid 'ssid_idx=0 ssid=SSID_USED security=SECURITY_USED password=SSID_PW_USED bssid=BSSID'",
                    " --ssid 'ssid_idx=1 ssid=SSID_USED security=SECURITY_USED password=SSID_PW_USED bssid=BSSID'",
                    " --sw_version DUT_SW --hw_version DUT_HW --serial_num DUT_SERIAL --model_num DUT_NAME"
                ]
            },
            "create_chamberview_wc":{
                "enabled":"TRUE",
                "load_db":"skip",
                "command":"create_chamberview.py",
                "args":"",
                "args_list":[
                    " --lfmgr LF_MGR_IP --port LF_MGR_PORT --delete_scenario",
                    " --create_scenario scenario_wpa2_wc",
                    " --raw_line \"profile_link 1.1 STA-AC 19 'DUT: DUT_NAME Radio-1' NA wiphy7,AUTO -1 NA\" ",
                    " --raw_line \"profile_link 1.1 upstream-dhcp 1 NA NA UPSTREAM_PORT,AUTO -1 NA\""
                ]
            },
            "wifi_capacity":{
                "enabled":"TRUE",
                "timeout":"600",
                "iterations":"1",
                "load_db":"skip",
                "command":"lf_wifi_capacity_test.py",
                "args":"",
                "args_list":[
                    " --mgr LF_MGR_IP --port LF_MGR_PORT --lf_user LF_MGR_USER --lf_password LF_MGR_PASS --instance_name scena
                    " --upstream UPSTREAM_PORT --batch_size 1,10,19 --loop_iter 1 --protocol UDP-IPv4 --duration 6000",
                    " --pull_report --local_lf_report_dir REPORT_PATH --test_tag 'wpa2_wc'",
                    " --test_rig TEST_RIG",
                    " --set DUT_SET_NAME"
                ]
            },
            "lf_qa":{
                "enabled":"TRUE",
                "timeout":"600",
                "load_db":"skip",
                "command":"./tools/lf_qa.py",
                "args":"",
                "args_list":[
                    " --path REPORT_PATH --store --png --database DATABASE_SQLITE"
                ]
            }
        }
    }
}
```

E. **sample command with above data:**

```
./lf_check.py  --json_rig ct_test_rig.json \
       --json_dut ct_AX88U_dut.json \
       --json_test ct_tests.json \
       --suite "suite_wc" \
       --path '/home/lanforge/html-reports/ct_results_directory'
```

4. **Set Up The JSON Configuration Files**

   A. The lf_check.py is run form the lanforge-scripts/py-scripts/tools directory

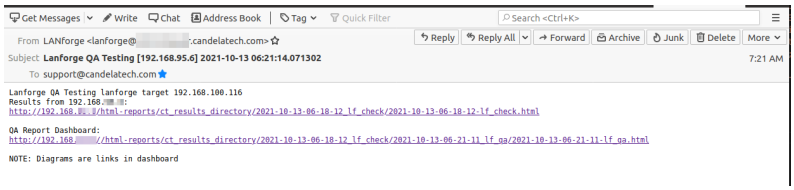
   B. lf_check.py uses three JSON files as input:
   For Example:
   ct_test_rig.json - describes the LANforge test rig configuration
   ct_AX88U_dut.json - describes the device under test
   ct_tests.json -describe the tests to be run.

5. **lf_check.py execution, simple command example**

   A.

```
./lf_check.py  --json_rig ct_test_rig.json \
       --json_dut ct_AX88U_dut.json \
       --json_test ct_tests.json \
       --suite "suite_wc" \
       --path '/home/lanforge/html-reports/ct_results_directory'
```

6. **Sample email sent on run**



7. **lf_check.py:** sample lf_check.py Report

LF Check: CT-TEST-001 lf_check.py

2021-10-13-06-18-12

Candela TECHNOLOGIES

Objective

Run QA Tests

LANforge

LANforge	kernel version	server version	gui version	gui build date	gui git sha	scripts git sha
ct523c-3b7c	5.13.0-rc5+	Version: 5.4.4 Compiled on: Mon 11 Oct 2021 05:51:28 PM PDT	5.4.4	"Mon 11 Oct 2021 06:39:52 PM PDT"	"e7462c3107bd9b139fb240edc0ab7437871bcb319"	e9888fd23d0cae429de7b877b86661e408ec7560\n

LANforge Radios

Radio	WIFI-Radio Driver	Radio Capabilities	Firmware Version	max_sta	max_vap	max_vifs
1.1.wiphy0	ath10k(9984)	802.11bgn-AC	10.4b-ct-9984-xtH-13-774502ee5	128	24	64
1.1.wiphy1	ath10k(9984)	802.11an-AC	10.4b-ct-9984-xtH-13-774502ee5	128	24	64
1.1.wiphy2	ath9k(i)	802.11abgn	<ath9k radios lack firmware>	2048	32	2048
1.1.wiphy3	ath10k(988x)	802.11abgn-AC	10.1-ct-8x__xtH-022-bcdb24ff	127	24	64
1.1.wiphy4	iwlwifi(AX200)	802.11abgn-AX	release/core62::5ecbd6da	1	1	1
1.1.wiphy5	iwlwifi(AX210)	802.11abgn-AX	release/core62::5ecbd6da	1	1	1
1.1.wiphy6	iwlwifi(AX210)	802.11abgn-AX	release/core62::5ecbd6da	1	1	1
1.1.wiphy7	mt7915e()	802.11abgn-AX	<no firmware data>	19	16	19

LF Check Test Results

Test	Command	Duration	Start	End	Result	STDOUT	STDERR
create_chamberview_dut_wc	./create_chamberview_dut.py	0d 2s 825293 ms	2021-10-13-06-18-16	2021-10-13-06-18-19	SUCCESS	STDOUT	
./create_chamberview_dut.py '--lfmgr 192.168.100.116 --port 8080 --dut_name ASUSRT-AX88U --sw1d 'sw1d_id=0 sw1d=asus11ax-2 security=wpa2 passwd=hel1o123 bssid=9c:7c:3f:55:4d:04' --sw1d 'sw1d_id=1 sw1d=asus11ax-5 security=wpa2 passwd=hel1o123 bssid=9c:7c:3f:55:4d:04' --sw_version DUT_SW_NA --hw_version DUT_HW_NA --serial_num DUT_SERIAL_NA --model_num ASUSRT-AX88U						
create_chamberview_wc	./create_chamberview.py	0d 5s 987380 ms	2021-10-13-06-18-19	2021-10-13-06-18-25	SUCCESS	STDOUT	
./create_chamberview.py '--lfmgr 192.168.100.116 --port 8080 --delete_scenario --create_scenario scenario1_wpa2_wc --sw_line "profile_link 1.1 STA-AC 19 'DUT: ASUSRT-AX88U Radio-2' NA wiphy7,AUTO -1 NA" --sw_line "profile_link 1.1 upstream-dhcp 1 NA NA 1.1,eth2,AUTO -1 NA"						
wifi_capacity	./lf_wifi_capacity_test.py	0d 16s 770807 ms	2021-10-13-06-18-25	2021-10-13-06-21-11	SUCCESS	STDOUT	
./lf_wifi_capacity_test.py '--mgr 192.168.100.116 --port 8080 --lf_user lanforge --lf_password lanforge --instance_name scenario_wpa2_wc --upstream 1.1.eth2 --batch_size 1,10,19 --loop_iter 1 --protocol UDP-IPv4 --duration 6000 --pull_report --local_lf_report_dir /home/lanforge/html-reports/ct_results_directory /2021-10-13-06-18-12_lf_check --test_tag 'wpa2_wc' --test_rig CT-TEST-001 --set DUT_NAME ASUSRT-AX88U'						
lf_qa	././tools/lf_qa.py	0d 2s 220803 ms	2021-10-13-06-21-11	2021-10-13-06-21-13	SUCCESS	STDOUT	
././tools/lf_qa.py '--path /home/lanforge/html-reports/ct_results_directory/2021-10-13-06-18-12_lf_check --store --png --database ./tools/qa_sqlite3.db'							

Generated by Candela Technologies LANforge network testing tool

www.candelatech.com

Candela TECHNOLOGIES

8. **lf_qa.py**
   lf_qa.py: process kpi.csv, produces html/pdf results, produces plotly png and interactive graphs from test run kpi
   sample command:

```
./lf_qa.py  --path  /home/lanforge/html-reports/ct_results_directory/(results dir of lf_check.py)\
  --store  \
  --png \
  --database ./tools/qa_aqlite3.db
```

9. **lf_qa.py:** sample lf_qa.py Report

LF QA: Verification Test Run

2021-10-13-06-21-11

Candela TECHNOLOGIES

Objective

QA Verification

Device Under Test

DUT	SW version	HW version	SN
ASUSRT-AX88U	DUT_SW_NA	DUT_HW_NA	NA

Test Rig: CT-TEST-001 Links

PDF Report
Current Test Suite Results Directory
All Test-Rig Test Suites Results Directory

Test Suite

Test	Test_Tag	Links
WiFi Capacity	wpa2_wc	html / pdf

Suite Summary

WiFi Capacity wpa2_wc	

QA Test Results

WiFi Capacity, Per Stations Rate UL+DL, wpa2_wc, CT-TEST-001, Report

WiFi Capacity, Per Stations Rate DL, wpa2_wc, CT-TEST-001, Report

10. **Sample lf_heck.py Output** example lf_check Report
11. **Test Control Inputs in Test Suite JSON**

A. `"enabled":"TRUE"`

Allows for individual test enable and disable of the test.

B. `"load_db":"CUSTOM_DATABASE"`

Allows for loading a LANforge database prior to the test run.

C. `"timeout":"300"`

Allows for test to have individual timeout other then default.

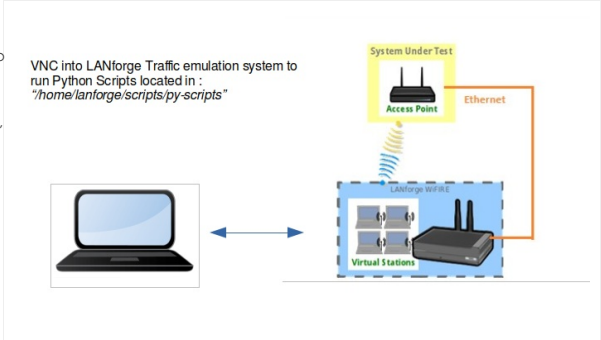
D. `"iterations":"2"`

Allows for test to run multiple iterations.

Start Here: Introduction to Executing Python Script on LANforge

Goal: Run First Python Script on LANforge

Each LANforge system has Python scripts preinstalled at `/home/lanforge/scripts` to configure the LANforge and run Traffic Emulation. Goal is to execute sta_connect2.py, one of the pre-installed python scripts located at

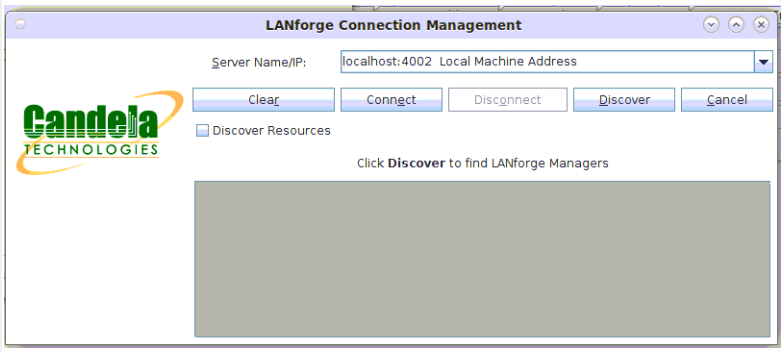


`/home/lanforge/scripts/py-script/sta_connect2.py`

The Script sta_connect2.py will create a station, create TCP and UDP traffic, run traffic for a short amount of time, and verify whether traffic was sent and received. It also verifies the station connected to the requested BSSID if bssid is specified as an argument. The script will clean up the station and connections at the end of the test. An html and pdf or the results will be generated and placed in /home/lanforge/html-reports directory The script will clean up the station and connections at the end of the test.

---

1. **Start the LANforgeGUI if GUI not running:**
   To start the LANforgeGUI navigate to : `/home/lanforge/LANforgeGUI_5.4.5`
   Execute : `./lfclient.bash`
   Select 'Connect' to connect to: `localhost:4002 Local Machine Address`



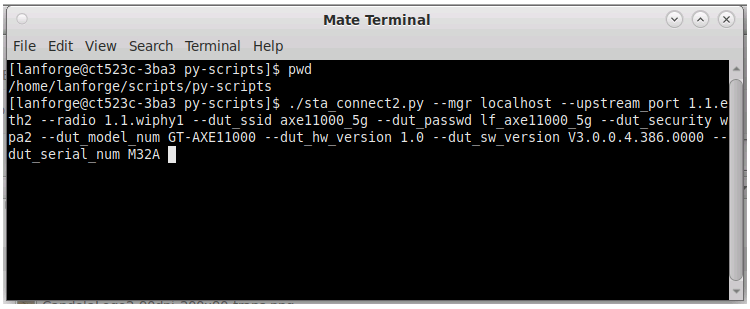
2. **Where Do I Find Scripts?**
   Preinstalled Python Scripts Location on LANforge: `/home/lanforge/scripts/py-scripts`
   Example script sta_connect2.py location: `/home/lanforge/scripts/py-scripts/sta_connect2.py`

3. **Initial Information to gather as input to sta_connect2.py script:**
   Note: An example of a Device Under Test is an Access Point
   The DUT information is used in report generation. The DUT information may be optional.

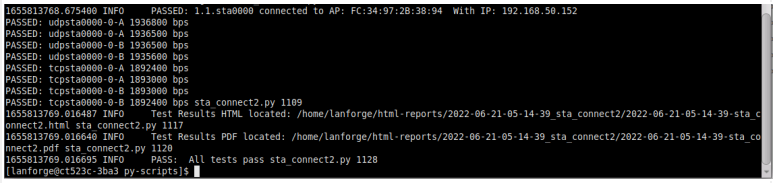
   A. The LANforge manager IP address: --mgr [localhost]
   B. The LANforge upstream port: --upstream_port [eth port]
   C. The LANforge radio: --radio [radio]
   D. The Device Under Test ssid: --dut_ssid [ssid]
   E. The Device Under Test passwd: --dut_passwd [passwd]
   F. The Device Under Test security: --dut_security [security]
   G. The Device Under Test Model Number: --dut_model_num [model]
   H. The Device Under Test Hardware Version: --dut_hw_version [hw version]
   I. The Device Under Test Software Version: --dut_sw_version [sw version]
   J. The Device Under Test Serial Number: --dut_serial_num [serial number]

4. **Example Command for sta_connect2.py:**

```
./sta_connect2.py --mgr localhost --upstream_port 1.1.eth2
    --radio 1.1.wiphy1 --dut_ssid axe11000_5g --dut_passwd lf_axe11000_5g --dut_security wpa2
    --dut_model_num GT-AXE11000 --dut_hw_version 1.0 --dut_sw_version V3.0.0.4.386.0000
    --dut_serial_num M32A
```



5. **Results for sta_connect2.py located in /home/lanforge/html-reports:**

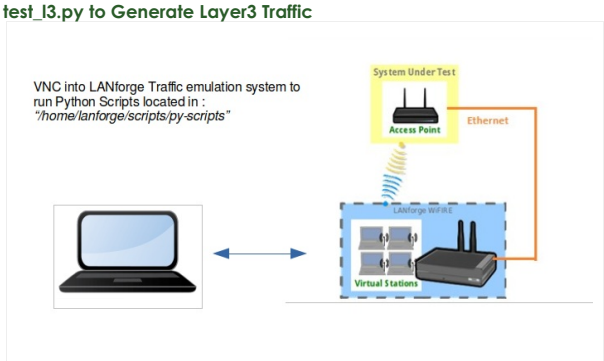


6. **Results for sta_connect2.py located in /home/lanforge/html-reports:**
   Script produces both html and pdf results
   Sample sta_connect2.py Script HTML Output: example of html output
   Sample sta_connect2.py Script pdf Output: example of pdf output
   Other script options may be shown by typing `./sta_connect2.py --help`

Basic: Layer3 Traffic Generation: test_l3.py

Goal: Use Python Script test_l3.py to Generate Layer3 Traffic

Each LANforge system has Python scripts installed at `/home/lanforge/scripts`. You can find it at



`/home/lanforge/scripts/py-script/test_l3.py`

The script **test_l3.py** will:

- create stations (on multiple radios),
- create TCP and UDP cross connects
- run traffic at specified data rates for a specified time.

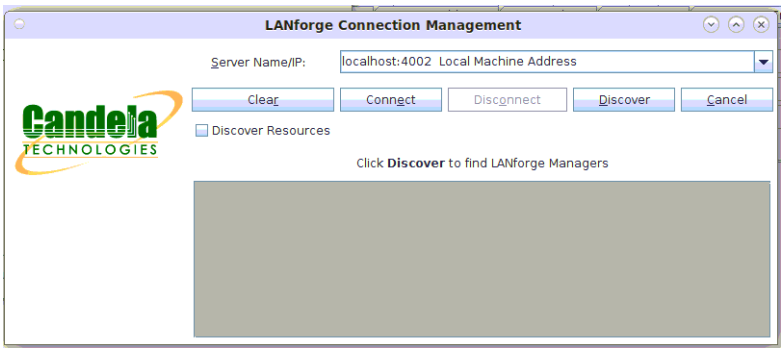
The traffic prioritization is configurable:

`BE`	Best Effort
`BK`	Background
`VI`	Video
`VO`	Video

The upload and download statistsics are recorded at the end of each polling interval. The script will verify whether traffic is sent and received. The script cleans up the station and connections at the end of the test. An HTML and PDF report of the results will be generated and placed in the `/home/lanforge/html-reports` directory.

---

1. **Start the LANforgeGUI if GUI not running:**
   To start the LANforgeGUI navigate to : /home/lanforge/LANforgeGUI_5.4.5
   Execute : `./lfclient.bash`
   Click the Connect button to connect to: `localhost:4002 Local Machine Address`



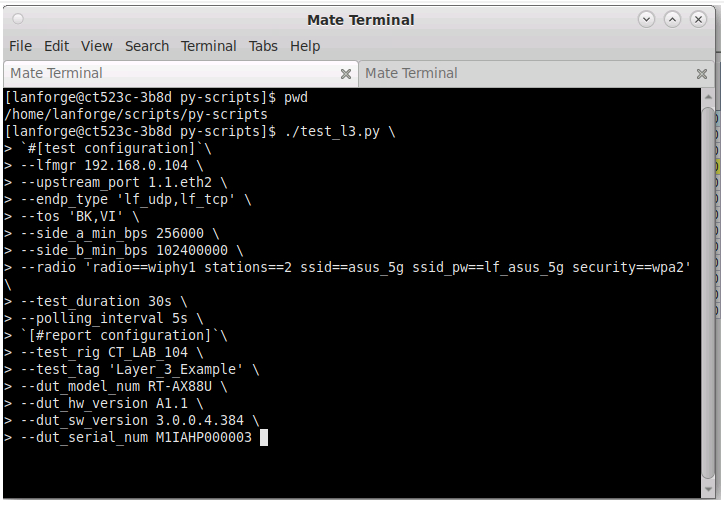
2. **Where Do I Find Scripts?**
   Preinstalled Python Scripts Location on LANforge: `/home/lanforge/scripts/py-scripts`
   Example script test_l3.py location: `/home/lanforge/scripts/py-scripts/test_l3.py`

3. **Initial Information to Gather as input to test_l3.py script:**
   Note: An example of a Device Under Test is an Access Point. The DUT information is used in report generation. The DUT information may be optional.

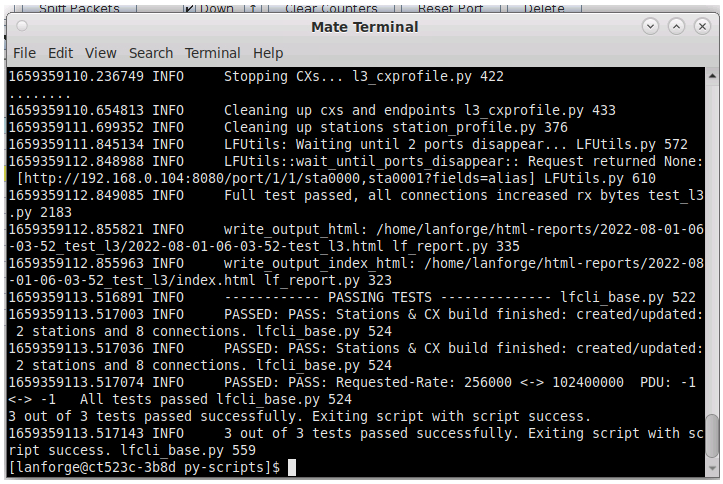
   A. The LANforge manager IP address: `--lfmgr [localhost]`
   B. The LANforge upstream port: `--upstream_port [eth port]`
   C. The LANforge end point type: `--endp_type 'lf_udp,lf_tcp'`
   D. The LANforge type of service: `--tos 'BK,VI'`
   E. The side 'a' tx bit rate (upload) `--side_a_min_bps [bits per second]`
   F. The side 'b' tx bit rate (download) `--side_b_min_bps [bits per second]`
   G. The LANforge radio information :
   `--radio 'radio==[radio] stations==[number] ssid==[ssid] ssid_pw==[password] security==[security]'`
   H. The Test Durations : `--test_duration [value]` (**s** - seconds, **m** - minutes, **h** - hours)
   I. The Polling Interval : `--polling_interval [value]` (**s** - seconds, **m** - minutes, **h** - hours)
   J. The Test Rig: `--test_rig [test system id]`
   K. The Test Tag: `--test_tag [unique test id]`
   L. The Device Under Test Model Number: `--dut_model_num [model]`
   M. The Device Under Test Hardware Version: `--dut_hw_version [hw version]`
   N. The Device Under Test Software Version: `--dut_sw_version [sw version]`
   O. The Device Under Test Serial Number: `--dut_serial_num [serial number]`

4. **Example Command for test_l3.py:**

```
./test_l3.py --lfmgr 192.168.0.103 \
--upstream_port 1.1.eth2 \
--endp_type 'lf_udp,lf_tcp' \
--tos 'BK,VI' \
--side_a_min_bps 256000 \
--side_b_min_bps 102400000 \
--radio 'radio==wiphy1 stations==1 ssid==asus_5g ssid_pw==lf_asus_5g security==wpa2' \
--test_duration 30s \
--polling_interval 5s \
--test_rig CT_LAB_104 \
--test_tag Layer_3_Example \
--dut_model_num RT-AX88U \
--dut_hw_version A1.1
--dut_sw_version 3.0.0.4.384 \
--dut_serial_num M1IAHP000003
```



5. **Results for test_l3.py located in `/home/lanforge/html-reports`:**



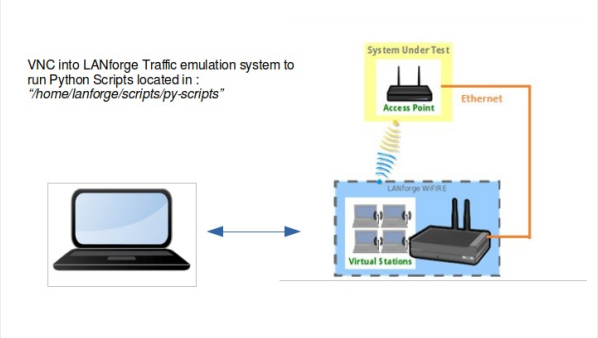
6. **Results for test_l3.py is located at /home/lanforge/html-reports:**
   Script produces both HTML and PDF results:
   - **example of HTML output**
   - **example of PDF output**
   - **example of kpi.csv output**

   Other script options may be show by typing `./test_l3.py --help`

Basic: Layer4 HTTP Traffic Generation: test_l4.py

Goal: Use Python Script test_l4.py to Generate Layer4 HTTP Traffic

Each LANforge system has Python scripts installed at `/home/lanforge/scripts`. You can find test_l4.py at



`/home/lanforge/scripts/py-script/test_l4.py`

The script **test_l4.py** will:

- Create stations (on the specified radio).
- Create Layer 4-7 endpoints.
- Monitor the bytes-rd attribute of the created endpoints.

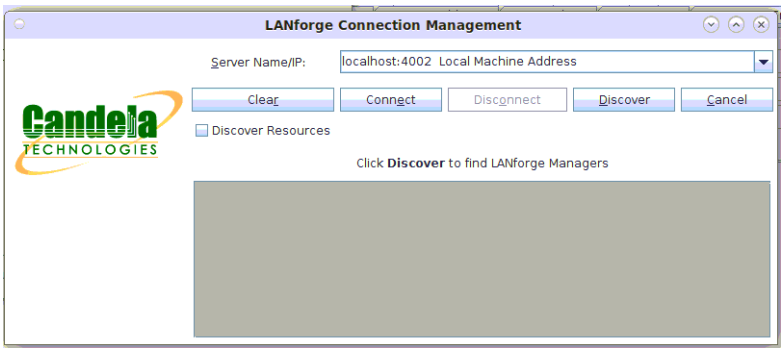
The test type attribute is configurable:

`bytes-rd`	monitor the bytes read
`urls`	monitor the url's per second

The monitored Layer 4-7 attribute statistics are recorded at the end of each polling interval. Test_l4.py will monitor the urls/s, bytes-rd, or bytes-wr attribute of the layer 4-7 endpoints. These attributes can be tested over FTP using a --ftp flag. If the monitored value does not continually increase, this test will not pass. The script cleans up the stations and connections at the end of the test. An HTML and PDF report of the results will be generated and placed in the `/home/lanforge/html-reports` directory.

---

1. **Start the LANforgeGUI if the GUI is not running:**
   To start the LANforgeGUI navigate to : /home/lanforge/LANforgeGUI_5.4.5
   Execute : `./lfclient.bash`
   Click the Connect button to connect to: `localhost:4002 Local Machine Address`



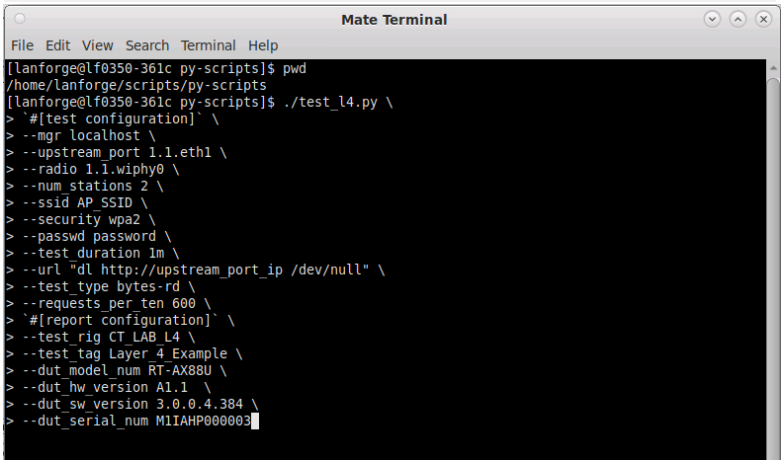
2. **Where Do I Find Scripts?**
   Preinstalled Python Scripts Location on LANforge: `/home/lanforge/scripts/py-scripts`
   Example script test_l4.py location: `/home/lanforge/scripts/py-scripts/test_l4.py`

3. **Initial Information to Gather as input for the test_l4.py script:**
   Note: An example of a Device Under Test is an Access Point. The DUT information is used in report generation. The DUT information may be optional.

   A. The LANforge manager IP address: `--mgr [localhost]`
   B. The LANforge upstream port: `--upstream_port [eth port]`
   C. The LANforge radio information : `--radio 'radio==[radio] stations==[number] ssid==[ssid] ssid_pw==[password] security==[security]'`
   D. The LANforge station creation amount: `--num_stations [2]`
   E. The AP SSID name: `--ssid [SSID]`
   F. The AP security type: `--security [open, wpa, wpa2, wpa3]`
   G. The AP SSID password: `--passwd [password]`
   H. The Test Duration : `--test_duration [value]` (**s** - seconds, **m** - minutes, **h** - hours)
   I. The Test URL: `--url "dl http://upstream_port_ip /dev/null"`
   J. The Test Type: `--test_type [bytes-rd, urls]`
   K. The Service Request Interval: `--requests_per_ten [600]`
   L. The Test Rig: `--test_rig [test system id]`
   M. The Test Tag: `--test_tag [unique test id]`
   N. The Device Under Test Model Number: `--dut_model_num [model]`
   O. The Device Under Test Hardware Version: `--dut_hw_version [hw version]`
   P. The Device Under Test Software Version: `--dut_sw_version [sw version]`
   Q. The Device Under Test Serial Number: `--dut_serial_num [serial number]`

4. **Example Command for a downloaded bytes-rd HTTP test with test_l4.py:**

```
./test_l4.py --lfmgr localhost \
--upstream_port 1.1.eth1 \
--radio 1.1.wiphy0 \
--num_stations 2 \
--ssid AP_SSID \
--security wpa2 \
--passwd password \
--test_duration 1m \
--url "dl http://upstream_port_ip /dev/null" \
--test_type bytes-rd \
--requests_per_ten 600 \
--test_rig CT_LAB_L4 \
--test_tag Layer_4_Example \
--dut_model_num RT-AX88U \
--dut_hw_version A1.1 \
--dut_sw_version 3.0.0.4.384 \
--dut_serial_num M1IAHP000003
```



5. **Example Command for a downloaded url's/s HTTP test with test_l4.py:**

```
./test_l4.py --lfmgr localhost \
--upstream_port 1.1.eth1 \
--radio 1.1.wiphy0 \
--num_stations 2 \
--ssid AP_SSID \
--security wpa2 \
--passwd password \
```

```
--test_duration 1m \
--url "dl http://upstream_port_ip /dev/null" \
--test_type urls \
--requests_per_ten 600 \
--test_rig CT_LAB_L4 \
--test_tag Layer_4_Example \
--dut_model_num RT-AX88U \
--dut_hw_version A1.1 \
--dut_sw_version 3.0.0.4.384 \
--dut_serial_num M1IAHP000003
```

Mate Terminal

File Edit View Search Terminal Help

```
[lanforge@lf0350-361c py-scripts]$ pwd
/home/lanforge/scripts/py-scripts
[lanforge@lf0350-361c py-scripts]$ ./test_l4.py \
> `#[test configuration]` \
> --lfmgr localhost \
> --upstream_port 1.1.eth1 \
> --radio 1.1.wiphy0 \
> --num_stations 2 \
> --ssid AP_SSID \
> --security wpa2 \
> --passwd password \
> --test_duration 1m \
> --url "dl http://upstream_port_ip /dev/null" \
> --test_type urls \
> --requests_per_ten 600 \
> `#[report configuration]` \
> --test_rig CT_LAB_L4 \
> --test_tag Layer_4_Example \
> --dut_model_num RT-AX88U \
> --dut_hw_version A1.1 \
> --dut_sw_version 3.0.0.4.384 \
> --dut_serial_num M1IAHP000003
```

6. Results for test_l4.py are located in `/home/lanforge/html-reports`:

Mate Terminal

File Edit View Search Terminal Help

```
7
1659551704.658922 INFO     item sta0000_l4 test_l4.py 256
1659551704.659637 INFO     item sta0001_l4 test_l4.py 256
1659551704.661618 INFO     self.csv_results_file -results.csv test_l4.py 216
1659551704.662043 INFO     csv_results_file: -results.csv test_l4.py 810
1659551704.727016 INFO     write_output_html: /home/lanforge/html-reports/2022-08-03-11-32-
25_test_l4/2022-08-03-11-32-25-test_l4.html lf_report.py 335
1659551704.728430 INFO     write_output_index_html: /home/lanforge/html-reports/2022-08-03-
11-32-25_test_l4/index.html lf_report.py 323
1659551706.899609 INFO     Stopping CXs... l4_cxprofile.py 71
..
1659551707.116932 INFO     Cleaning up cxs and endpoints l4_cxprofile.py 131
1659551707.358503 INFO     Cleaning up stations station_profile.py 376
1659551707.584649 INFO     LFUtils: Waiting until 2 ports disappear... LFUtils.py 572
1659551709.141561 INFO     LFUtils::wait_until_ports_disappear:: Request returned None: [ht
tp://localhost:8080/port/1/1/sta0000,sta0001?fields=alias] LFUtils.py 610
1659551709.142251 INFO     LFUtils: Waiting until 2 ports disappear... LFUtils.py 572
1659551709.156003 INFO     LFUtils::wait_until_ports_disappear:: Request returned None: [ht
tp://localhost:8080/port/1/1/sta0000,sta0001?fields=alias] LFUtils.py 610
1659551709.157082 INFO     Full test passed test_l4.py 879
1659551709.157956 INFO     ------------ PASSING TESTS -------------- lfcli_base.py 522
1659551709.158563 INFO     PASSED: PASS: Station build finished lfcli_base.py 524
1659551709.158879 INFO     PASSED: All stations got IPs lfcli_base.py 524
2 out of 2 tests passed successfully. Exiting script with script success.
1659551709.159600 INFO     2 out of 2 tests passed successfully. Exiting script with script
 success. lfcli_base.py 559
[lanforge@lf0350-361c py-scripts]$
```

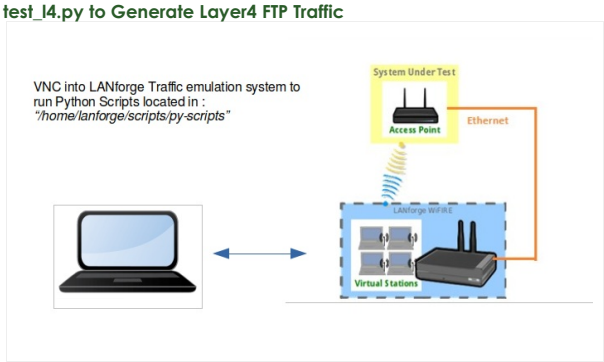
7. Results for test_l4.py are located in /home/lanforge/html-reports:
   The script produces both HTML and PDF results:
   - example of HTML output
   - example of PDF output
   - example of kpi.csv output

   Additional script options may be shown by typing `./test_l4.py --help`

Basic: Layer4 FTP Traffic Generation: test_l4.py

Goal: Use Python Script test_l4.py to Generate Layer4 FTP Traffic

Each LANforge system has Python scripts installed at `/home/lanforge/scripts`. You can find test_l4.py at



`/home/lanforge/scripts/py-script/test_l4.py`

The script **test_l4.py** will:

- Create stations (on the specified radio).
- Create Layer 4-7 endpoints.
- Monitor the bytes-rd attribute of the created endpoints.

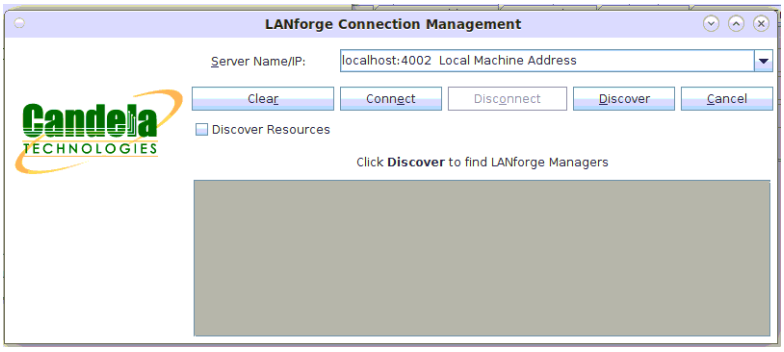
The test type attribute is configurable:

`bytes-rd`	monitor the bytes read
`bytes-wr`	monitor the bytes write
`urls`	monitor the url's per second

The monitored Layer 4-7 attribute statistics are recorded at the end of each polling interval. Test_l4.py will monitor the urls/s, bytes-rd, or bytes-wr attribute of the layer 4-7 endpoints. These attributes can be tested over FTP using a --ftp flag. If the monitored value does not continually increase, this test will not pass. The script cleans up the stations and connections at the end of the test. An HTML and PDF report of the results will be generated and placed in the `/home/lanforge/html-reports` directory.

---

1. **Start the LANforgeGUI if the GUI is not running:**
   To start the LANforgeGUI navigate to : /home/lanforge/LANforgeGUI_5.4.5
   Execute : `./lfclient.bash`
   Click the Connect button to connect to: `localhost:4002 Local Machine Address`



2. **Where Do I Find Scripts?**
   Preinstalled Python Scripts Location on LANforge: `/home/lanforge/scripts/py-scripts`
   Example script test_l4.py location: `/home/lanforge/scripts/py-scripts/test_l4.py`
3. **Initial Information to Gather as input for the test_l4.py script:**
   Note: An example of a Device Under Test is an Access Point. The DUT information is used in report generation. The DUT information may be optional.
   A. The LANforge manager IP address: `--mgr [localhost]`
   B. The LANforge upstream port: `--upstream_port [eth port]`
   C. The LANforge radio information : `--radio 'radio==[radio] stations==[number] ssid==[ssid] ssid_pw==[password] security==[security]'`
   D. The LANforge station creation amount: `--num_stations [2]`
   E. The AP SSID name: `--ssid [SSID]`
   F. The AP security type: `--security [open, wpa, wpa2, wpa3]`
   G. The AP SSID password: `--passwd [password]`
   H. The Test Duration : `--test_duration [value]` (**s** - seconds, **m** - minutes, **h** - hours)
   I. The FTP Test Switch: `--ftp [enables FTP testing]`
   J. The Test URL: `--url "dl ftp://upstream_port_ip /dev/null"`

K. The Test Type: `--test_type [bytes-rd]`

L. The Service Request Interval: `--requests_per_ten [600]`

M. The Test Rig: `--test_rig [test system id]`

N. The Test Tag: `--test_tag [unique test id]`

O. The Device Under Test Model Number: `--dut_model_num [model]`

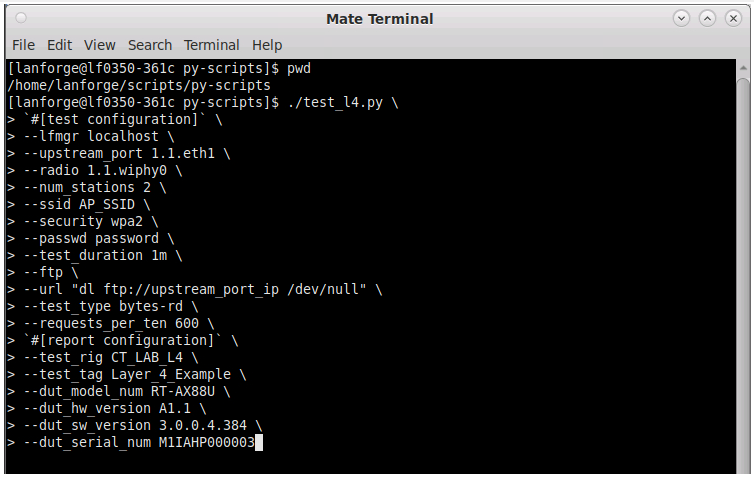
P. The Device Under Test Hardware Version: `--dut_hw_version [hw version]`

Q. The Device Under Test Software Version: `--dut_sw_version [sw version]`

R. The Device Under Test Serial Number: `--dut_serial_num [serial number]`

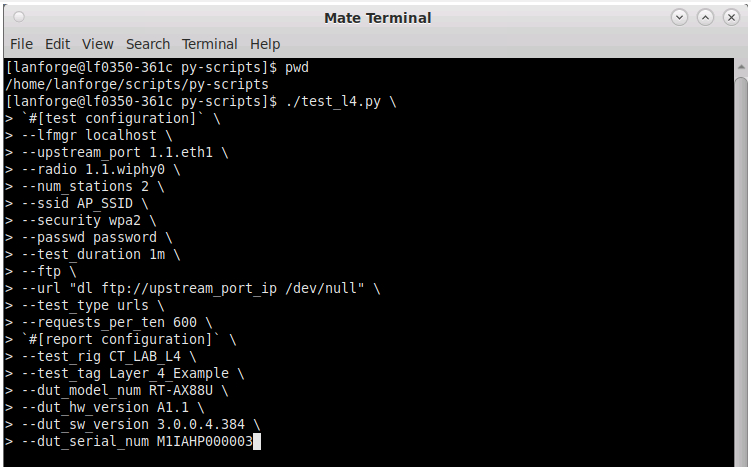
4. **Example Command for a downloaded bytes-rd FTP test with test_l4.py:**

```
./test_l4.py --lfmgr localhost \
--upstream_port 1.1.eth1 \
--radio 1.1.wiphy0 \
--num_stations 2 \
--ssid AP_SSID \
--security wpa2 \
--passwd password \
--test_duration 1m \
--ftp \
--url "dl ftp://upstream_port_ip /dev/null" \
--test_type bytes-rd \
--requests_per_ten 600 \
--test_rig CT_LAB_L4 \
--test_tag Layer_4_Example \
--dut_model_num RT-AX88U \
--dut_hw_version A1.1 \
--dut_sw_version 3.0.0.4.384 \
--dut_serial_num M1IAHP000003
```



5. **Example Command for a downloaded url's/s FTP test with test_l4.py:**

```
./test_l4.py --lfmgr localhost \
--upstream_port 1.1.eth1 \
--radio 1.1.wiphy0 \
--num_stations 2 \
--ssid AP_SSID \
--security wpa2 \
--passwd password \
--test_duration 1m \
--ftp \
--url "dl ftp://upstream_port_ip /dev/null" \
--test_type urls \
--requests_per_ten 600 \
--test_rig CT_LAB_L4 \
--test_tag Layer_4_Example \
--dut_model_num RT-AX88U \
--dut_hw_version A1.1 \
--dut_sw_version 3.0.0.4.384 \
--dut_serial_num M1IAHP000003
```



6. **Results for test_l4.py are located in `/home/lanforge/html-reports`:**

```
Mate Terminal
File  Edit  View  Search  Terminal  Help
7
1659719453.671809 INFO     item sta0000_l4 test_l4.py 256
1659719453.672634 INFO     item sta0001_l4 test_l4.py 256
1659719453.674645 INFO     self.csv_results_file -results.csv test_l4.py 216
1659719453.675257 INFO     csv_results_file: -results.csv test_l4.py 810
1659719453.741720 INFO     write_output_html: /home/lanforge/html-reports/2022-08-05-10-06-
15_test_l4/2022-08-05-10-06-15-test_l4.html lf_report.py 335
1659719453.743767 INFO     write_output_index_html: /home/lanforge/html-reports/2022-08-05-
10-06-15_test_l4/index.html lf_report.py 323
1659719455.664310 INFO     Stopping CXs... l4_cxprofile.py 71
..
1659719455.958650 INFO     Cleaning up cxs and endpoints l4_cxprofile.py 131
1659719456.176748 INFO     Cleaning up stations station_profile.py 376
1659719456.398688 INFO     LFUtils: Waiting until 2 ports disappear... LFUtils.py 572
1659719457.430698 INFO     LFUtils::wait_until_ports_disappear:: Request returned None: [ht
tp://localhost:8080/port/1/1/sta0000,sta0001?fields=alias] LFUtils.py 610
1659719457.431388 INFO     LFUtils: Waiting until 2 ports disappear... LFUtils.py 572
1659719457.443248 INFO     LFUtils::wait_until_ports_disappear:: Request returned None: [ht
tp://localhost:8080/port/1/1/sta0000,sta0001?fields=alias] LFUtils.py 610
1659719457.444070 INFO     Full test passed test_l4.py 879
1659719457.444743 INFO     ------------ PASSING TESTS -------------- lfcli_base.py 522
1659719457.445157 INFO     PASSED: PASS: Station build finished lfcli_base.py 524
1659719457.445669 INFO     PASSED: All stations got IPs lfcli_base.py 524
2 out of 2 tests passed successfully. Exiting script with script success.
1659719457.446345 INFO     2 out of 2 tests passed successfully. Exiting script with script
 success. lfcli_base.py 559
[lanforge@lf0350-361c py-scripts]$
```

7. **Results for test_l4.py are located in /home/lanforge/html-reports:**
   The script produces both HTML and PDF results:
   - **example of HTML output**
   - **example of PDF output**
   - **example of kpi.csv output**

   Additional script options may be shown by typing `./test_l4.py --help`

Customize and Run the Python Dataplane Script

Goal: Edit and run the bash script that runs both Python Dataplane and Create Chamberview scripts according to a customized LANforge Setup.

This cookbook describes how to edit a bash script, cv_dataplane_script.sh, that executes the 'Create Chamber DUT', 'Create Chamberview' and the 'Dataplane' test python scripts (create_chamberview_dut.py, create_chamberview.py, lf_dataplane_test.py). These 3 python scripts are broken up into sections within this one bash script, that have their own arguments passed into each python script. The python scripts will run in consecutive order within the bash script and the LANforge GUI will reflect when each python script runs. Requires LANforge 5.4.2.

---

1. **Open the bash script and edit the variables at the top of the script. Also, understand a little bit about this bash script.**
   A. This bash script, 'cv_dataplane_script.sh, is comprised of 3 different python scripts, broken up into their own sections. This cookbook explains how to run these 3 different python scripts.
   B. In this code, when executing a python script, the arguements are denoted by a '--' in front of them. This is then followed by the arguement name, a equals sign, and then the actual arguement (user input). The format: '--argument_name_1=[USER INPUT ARGUMENT 1] --argument_name_2=[USER INPUT ARGUEMENT 2]. Only 1 space must be between the end of an argument input and next arguement name. Never include a space between the equal sign and arguement/arguement name.

```
--station 1.01.wlan0
```

   C. Variables are denoted at the top in the format 'VARIABLE_NAME=YOUR_VALUE' When the variables are used in the script, the format is '${VARIABLE_NAME}'. More or less variables can be added and removed from the script if used in this format.

```
#!/bin/bash

## NOTES ##

# Define some common variables.  Change these to match the current testbed.
MGR=192.168.102.211
PORT=8080
DUT_NAME="TEST_DUT"
```

   D. Throughout the script, there are backslashes at the end of the line if the arguments are continuing over to the next line. The backslashes indicate the code to combine both lines together when running.

```
./create_chamberview_dut.py --lfmgr ${MGR} -o ${PORT} --dut_name ${DUT_NAME} --dut_flag="DHCPD-LAN" --dut_flag="DHCPD-WAN" \
--ssid "ssid_idx=0 ssid=eero-mesh-lanforge security=WPA2 password=lanforge bssid=64:97:14:64:D9:06" \
```

   E. Lastly, to see any further detail about scripts, run the script in the same format as the bash code: go to the py-scripts directory and run the --help arguement. In the create_chamberview_dut python script this would look like: './create_chamber_dut.py --help'. This --help argument gives more detail about the script.

```
[lanforge@ct523c-Scale-Mobsta ~]$ cd /home/lanforge/lanforge-scripts/py-scripts
[lanforge@ct523c-Scale-Mobsta py-scripts]$ ./create_chamberview_dut.py --help
```

2. **Create the DUT: edit the arguments to pass into create_chamberview_dut.py**

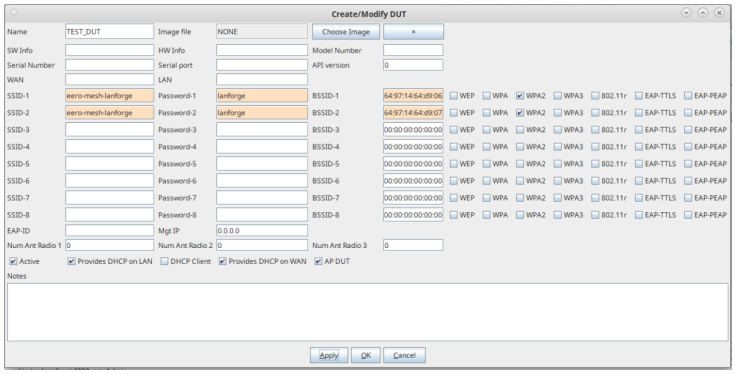
```
./create_chamberview_dut.py --lfmgr ${MGR} -o ${PORT} --dut_name ${DUT_NAME} --dut_flag="DHCPD-LAN" --dut_flag="DHCPD-WAN" \
--ssid "ssid_idx=0 ssid=eero-mesh-lanforge security=WPA2 password=lanforge bssid=64:97:14:64:D9:06" \
--ssid "ssid_idx=1 ssid=eero-mesh-lanforge security=WPA2 password=lanforge bssid=64:97:14:64:D9:07"
```

A. First, the argument names --lf_mgr, -o (port) and --dut_name are all passed in from the top (and they are all required). DUT Flags are flags used by the server, below is a screenshot of all the flags. To calculate the number for all the flags needed, t

dut_flags:

```
STA_MODE      | 0x1     # (1) DUT acts as Station.
AP_MODE       | 0x2     # (2) DUT acts as AP.
INACTIVE      | 0x4     # (3) Ignore this in ChamberView, etc
WEP           | 0x8     # Use WEP encryption on all ssids, deprecated, see add_dut_ssid.
WPA           | 0x10    # Use WPA encryption on all ssids, deprecated, see add_dut_ssid.
WPA2          | 0x20    # Use WPA2 encryption on all ssids, deprecated, see add_dut_ssid.
DHCPD-LAN     | 0x40    # Provides DHCP server on LAN port
DHCPD-WAN     | 0x80    # Provides DHCP server on WAN port
WPA3          | 0x100   # Use WPA3 encryption on all ssids, deprecated, see add_dut_extras.
11r           | 0x200   # Use .11r connection logic on all ssids, deprecated, see add_dut_ssid.
EAP-TTLS      | 0x400   # Use EAP-TTLS connection logic on all ssids, deprecated, see add_dut_ssid.
EAP-PEAP      | 0x800   # Use EAP-PEAP connection logic on all ssids, deprecated, see add_dut_ssid.
NOT-DHCPCD    | 0x1000  # Station/edge device that is NOT using DHCP.
                        # Otherwise, automation logic assumes it is using dhcp client.
```

B. Next, add the ssid lines. Each --ssid argument is followed by a string with several individual arguments. ssid_idx is the number which ssid it is. This number is just a sequential number, the first one is 1, second one is 2, etc. 'ssid' is ssid from the AP. This is the same for password, security, BSSID. Multiple securities example for a SSID is shown in the second --line for ssid_idx 2.



3. Create the Chamber View scenario: edit the flags to pass into create_chamberview.py

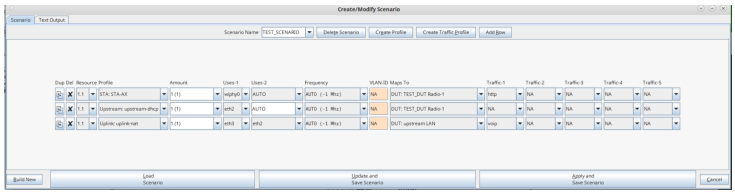
The image below highlights the section of the script to be edited.

```
# Create/update chamber view scenario and apply and build it.
echo "Build Chamber View Scenario"
#change the lfmgr to your system, set the radio to a working radio on your LANforge system, same with the ethernet port.
./create_chamberview.py --mgr ${MGR} --mgr_port ${PORT} --delete_scenario --create_scenario "TEST_SCENARIO" \
--line "Resource=1.1 Profile=STA-AX Amount=1 Uses-1=wiphy0 DUT="${DUT_NAME}" DUT_Radio=Radio-1 Traffic=http Freq=-1" \
--line "Resource=1.1 Profile=upstream-dhcp Amount=1 Uses-1=eth2 Traffic=NA Freq=-1 " \
--line "Resource=1.1 Profile=uplink-nat Amount=1 Uses-1=eth3 Uses-2=eth2 Traffic=voip DUT=upstream DUT_Radio=LAN Freq=-1"
```

A. As shown in the example, the first line comprises of the following flags: --mgr (lanforge IP address), --mgr_port (the port through which this script will use), --delete_scenario, and --cs (name under which this new scenario will be saved under in the database).The mgr and mgr_port are passed in through the variables at the top of this bash script and the scenario name can be anything. If the scenario name passed in as --create_scenario is already in the GUI, using the '--delete_scenario' flag will override that already created scenario.

```
./create_chamberview.py --mgr ${MGR} --mgr_port ${PORT} --delete_scenario --create_scenario "TEST_SCENARIO" \
```

B. The next arguments, '--line' are the lines that show up in the GUI if the scenario is created manually. These lines will translate in the GUI after the command is executed. After the '--line' is the actual line, a string. The string has details of the objects of the dataplane test (such as amount, radio, etc.), some required, some not. In the example is used a 'station' line, 'upstream-dhcp' line, and 'uplink-nat' line (as all these are objects in our dataplane test).1st, 'Resource' (required) is resource number the object is located on. The first number will most likely always be 1 (Shelf) and the second number is Resource (in this case, also 1). Notation for resource 4 would be '1.4'. Profile (required) is the name of the profile wanted for an object. Profiles can be created and found in the profiles tab in the GUI. The profile used in the example is 'STA-AX'(with the station profile type), for the station profile. 'Amount' (required) is the amount of objects to be created in chamber view. For stations, the amount can be multiple, for ethernet object creations it will most likely be 1.'Uses-1' (required) is the object this new created object will use or reside on. For an upstream object, eth1 through eth3 might be the 'Uses-1'. For station objects, this is the radio that the station will use. 'Uses-2' is optional and an alternative to 'Uses-1'.'Traffic' (optional) is background traffic that object runs and can be either voip, http and others found in the 'Traffic' dropdown of the Scenario Creation GUI.



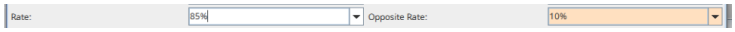
C. 'DUT' and 'DUT_Radio' are not optional. In the station line, '$DUT' is taking in the DUT name variable passed in at the top, but it can be any DUT name that is already created in the GUI. 'DUT_Radio corresponds with the SSID number in the DUT object. 'Radio-1' is corresponding to 'SSID-1'. For the upstream object, the 'Radio' is the 'LAN' port on the DUT. Finally, 'Freq' is the frequency the object's radio should be on. This mainly is for stations and objects that use radios.'

4. Edit the arguments to run the dataplane test.

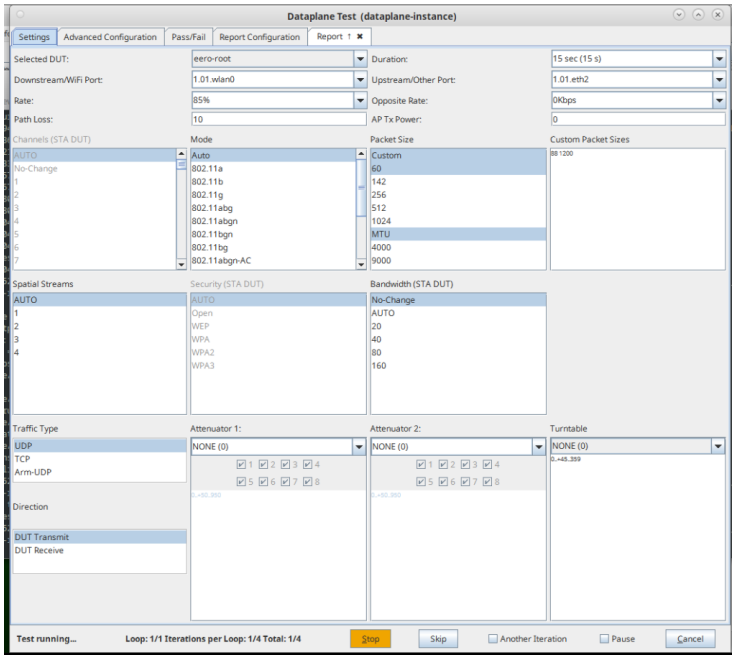
```
./lf_dataplane_test.py --mgr ${MGR} -o ${PORT} --instance_name dataplane-inst --upstream 1.01.eth2 --download_speed "85%" --upload_speed "10%" --station 1.01.wlan0 --dut ${DUT_NAME} \
--raw_line "pkts: Custom;60;MTU" --raw_line "cust_pkt_sz: 88 1200" --raw_line "directions: DUT Transmit" --raw_line "traffic_types:UDP"--raw_line --duration "30s"
```

A. Similar to other python scripts run in this bash script, '--mgr', '--o' (port) and '--dut' are passed in through the variables at the top. '--instance_name is the name of the new window that will hold the dataplane test.--upstream_port is more than likely an ethernet port, one used earier when creating the scenario. The example used below is eth3. This notation is the '[shelf].[resource].[name]' notation. The shelf and the resource can be found in the 'Port Manager', under the 'Port' column. The shelf and the resource are the first 2 numbers separated by the first dot. This same notation is used for the '--station' flag too. The 'station' flag is the station used in the dataplane test.

B. Upload and download speed are in percentages or Kbps/Mbps/Gbps. It is the requested connection traffic speed. If a percentage is entered, the rate will be calculated from the theoretical throughput.



C. The '--raw_line' flags are similar to those used earlier in other python scripts. They are permutations/combinations of the dataplane test, which will reflect in the window that pops up within the GUI. They all need to have the same format used as in the example (same spacing, units if need be).



5. **Run the bash script!**

```
[lanforge@ct523c-Scale-Mobsta py-scripts]$ ./cv_dataplane_script.sh
```

Multiplexed REST Access via Nginx Proxy

Goal: Configure an NGINX proxy to allow REST traffic to a variety of isolated LANforge machines

It is possible to configure a Nginx proxy in a manner to allow remote REST clients access to multiple isolated LANforge systems. This leverages the proxy_pass feature in Nginx. There are multiple ways to configure proxy access.

For the example below, we will assume these values:

- **public proxy hostname is bizproxy, 10.39.0.44**
- **bizproxy is running Nginx**
- **Isolated LAN with LF machines: 192.168.92.0/24**
- **Example LANforge machines:**
  - **192.168.92.10 ct523-jedway1**
  - **192.168.92.11 ct522-jedway3**
- **the LANforge machines need to have GUIs configured to start automatically**

LANforge GUI HTTP Processing

The HTTP library that the LANforge GUI incorporates is very simple. It is not configured to parse *Host:* headers. There is no need to rewrite the Host header when proxying to port 8080.

Proxying to Apache on LANforge (mgt_ip, port 80) is different. If you want to proxy requests to a LF Apache instance on port 80, you *should incorporate Host header rewriting*. (No examples below, sorry.)

Proxy Request Rewriting

Three ways of making proxy requests include:

- **Port Rewriting. Works best with our python libraries.**
- ***Hostname Rewriting,* more difficult, but still works with python libraries.**
- **URL (path-name) Rewriting: this does NOT work well with our python libraries.**

Port Rewriting

This manner of proxying just translates different server listening ports to the target machines. It is another easy transformation, but it opens up quite a number of high-numbered ports on bizproxy.
Nginx config:

```
server {
   listen 1910;
   server_name _;
   root /usr/share/nginx/html;

    location / {
       rewrite            /(.*) /$1 break;
       proxy_pass         http://192.168.92.10:8080;
       proxy_redirect     off;
       proxy_set_header   Host $host;
       proxy_set_header   X-Real-Ip $remote_addr;
       proxy_set_header   X-Forwarded-For $remote_addr;
    }
}
server {
   listen 1911;
   server_name _;
   root /usr/share/nginx/html;

    location / {
       rewrite            /(.*) /$1 break;
       proxy_pass         http://192.168.92.11:8080;
       proxy_redirect     off;
       proxy_set_header   Host $host;
       proxy_set_header   X-Real-Ip $remote_addr;
       proxy_set_header   X-Forwarded-For $remote_addr;
    }
}
```

Use curl to test access:

```
curl -sqv -H 'Accept: application/html' http://bizproxy:1910/port/1/1/list
```

Example script usage:

```
./scenario.py --mgr bizproxy --mgr_port 1910 \
    --load BLANK --action overwrite
```

Hostname Rewriting

It is possible to rewrite hostnames and host headers to isolated LF systems. This iscomplicated rewrite because the DNS names need to be present at the developer's workstation. (*It is unlikely that the the headers in the HTTP request can be manipulated to add the Host header.*) Ideally, the *non-isolated LAN* DNS can be configured to return the return the IP of bizproxy.corp.me when hostnames like ct523-jedway1.bizproxy.corp.me are requested.

On the developer workstation, this is possible with extra effort on the user side by manipulating the `/etc/hosts` file on a workstation:

```
# etc/hosts
10.39.0.44    ct523-jedway1.bizproxy.corp.me    ct523-jedway1
```

Nginx config:

```
server {
    listen 80;
    server_name ct523-jedway1;
    root /usr/share/nginx/html;

    location / {
        rewrite            /(.*) /$1 break;
        proxy_pass         http://192.168.92.10:8080;
        proxy_redirect     off;
        proxy_set_header   Host $host;
        proxy_set_header   X-Real-Ip $remote_addr;
        proxy_set_header   X-Forwarded-For $remote_addr;
    }
}
```

Check the URL access using curl:

```
# check by IP:
$ curl -sqv \
    -H 'Host: ct523-jedway1' \
    -H 'Accept: application/json' \
    http://10.39.0.44/port/1/1/list

# check by hostname
$ curl -sqv \
    -H 'Accept: application/json' \
    http://ct523-jedway1.bizproxy.corp.me/port/1/1/list
```

Example script usage:

```
./scenario.py --mgr ct523-jedway1 --mgr_port 80 \
   --load BLANK --action overwrite
```

Logging HTTP Access

The bizproxy logs should be located in `/var/log/nginx`. In LF 5.4.6, the GUI can send messages to syslog.

Messages from the GUI would look like:

```
1685573102952:  ip[192.168.92.1] sess[] GET url[/port/1/1/list]
```

Appendix

URL Rewriting is mentioned here so the reader can understand what *not* to configure.

URL Rewriting

Below is an example permitting REST access to LF hosts by way of a URL prefix. For example, the URL http://bizproxy/92.11/port/1/1/list becomes the URL http://192.168.92.11:8080/port/1/1/list . This is not the best kind of proxy rewriting, but it is the easiest. Using a URL prefix is less ideal because it inherently conflicts with the LANforge python libraries provided.
Nginx config:

```
server {
    listen       80;
    server_name  _;
    root         /usr/share/nginx/html;

    # Load configuration files for the default server block.
    include /etc/nginx/default.d/*.conf;

    location /92.10 {
       rewrite            /92.10/(.*) /$1 break;
       proxy_pass         http://192.168.92.10:8080;
       proxy_redirect     off;
       proxy_set_header   Host biz_lflab5_9210;
       proxy_set_header   X-Real-Ip $remote_addr;
       proxy_set_header   X-Forwarded-For $remote_addr;
    }
    location /92.11 {
       rewrite            /92.11/(.*) /$1 break;
       proxy_pass         http://192.168.92.11:8080;
       proxy_redirect     off;
       proxy_set_header   Host $host;
       proxy_set_header   X-Real-Ip $remote_addr;
       proxy_set_header   X-Forwarded-For $remote_addr;
    }
}
```

Use curl to query the REST endpoint:

```
$ curl -sqv -H 'Accept: application/json' http://bizproxy/92.10/port/1/1/list
```

This is not compatible with the py-scripts library.