Candela TECHNOLOGIES Network Testing and Emulation Solutions

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

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**

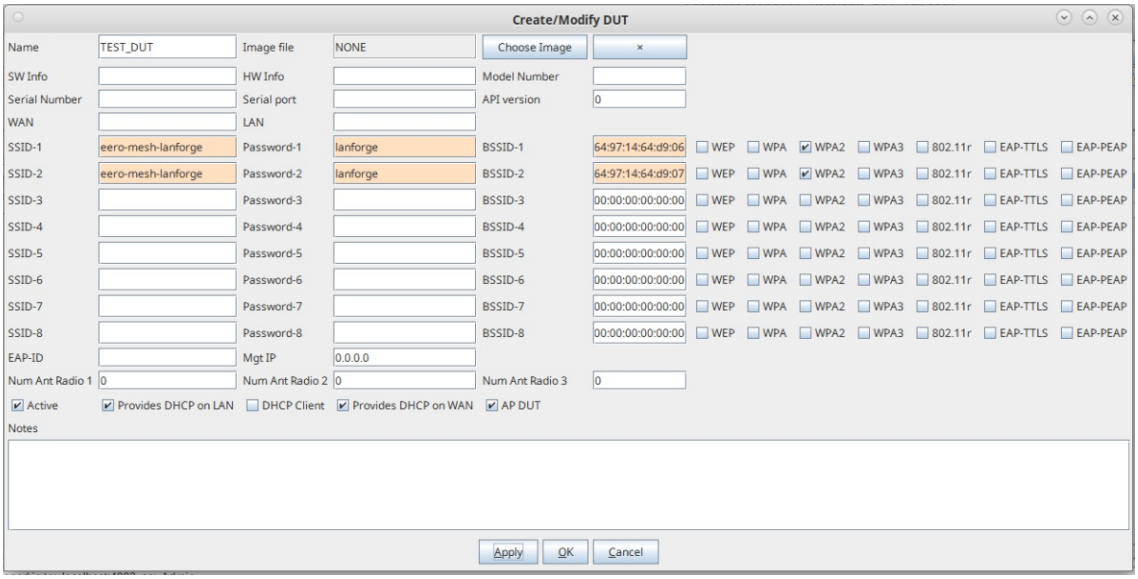
   ```
   ./create_chamberview_dut.py --lfmgr ${MGR} -o ${PORT} --dut_name ${DUT_NAME} --dut_flag="DHCPD-LAN" --dut_flag="DHCPD-WAN" \
   --ssid "ssid_idx=0 ssid=eero-mesh-lanforge security=WPA2 password=lanforge bssid=64:97:14:64:D9:06" \
   --ssid "ssid_idx=1 ssid=eero-mesh-lanforge security=WPA2 password=lanforge 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

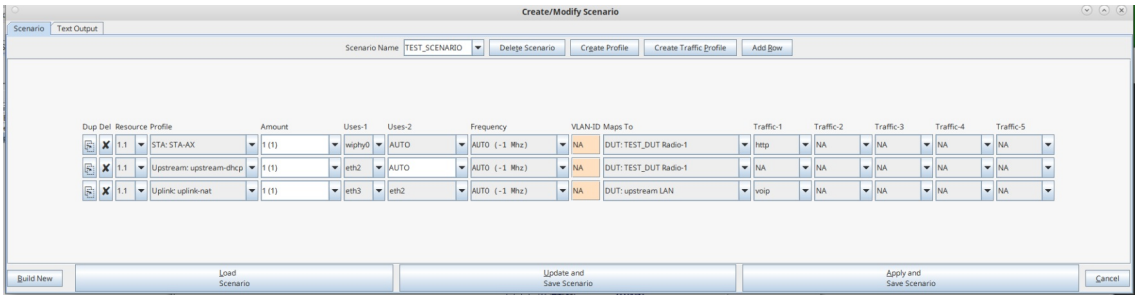
The image below highlights the section of the script to be edited.

```
# 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.



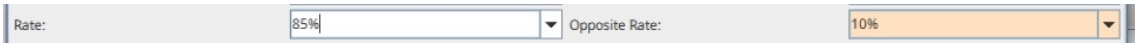
C. 'DUT' and 'DUT_Radio' are not optional. In the station line, '$DUT' is taking in the DUT name variable passed in at the top, but it can be any DUT name that is already created in the GUI. 'DUT_Radio corresponds with the SSID number in the DUT object. 'Radio-1' is corresponding to 'SSID-1'. For the 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.**

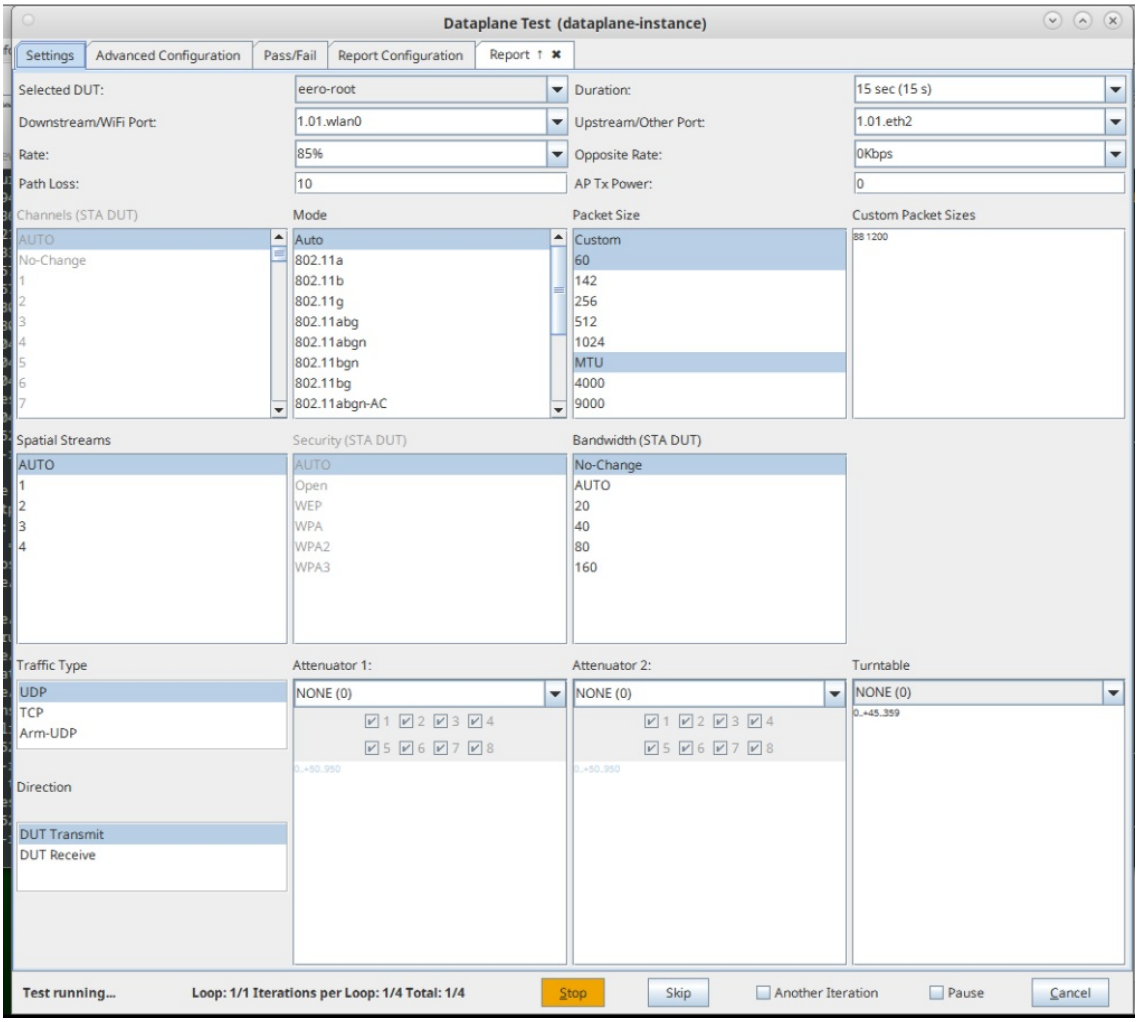
```
./lf_dataplane_test.py --mgr ${MGR} -o ${PORT} --instance_name dataplane-inst --upstream 1.01.eth2 --download_speed "85%" --upload_speed "10%" --station 1.01.wlan0 --dut {$DUT_NAME} \
--raw_line "pkts: Custom;60;MTU" --raw_line "cust_pkt_sz: 88 1200" --raw_line "directions: DUT Transmit" --raw_line "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
```