ChaosLemur API v0.1
Dec. 6th, 2015

Emmanuel Shiferaw
Davis Gossage

class ChaosLemurConfigGenerator:

Bold/Highlighted = public-facing

ChaosLemurConfigGenerator contains methods used to set up the ChaosLemur environment, for experiments to be run. It takes as input all of the customizable/parameterized inputs for the environment, including number of routers, topology, distribution pattern to use for number of networks advertised, and the parameters (with default/optional values) for those distributions.

```
    DEFAULT_AS = 7675
    def __init__(self, num_routers, topology, net_distrib, dist_param_1=1,
dist_param_2=10):

    # Load bgpd.conf.template file from fs, extract portion describing self/neighbors
    def loadTemplate(self):

    # Given distribution pattern specified, calculate number of subnets initially
loaded into each router.
    def calculateDistributions(self):

    # Given desired number of routers and topology, will generate
    # bgpd.conf quagga BGP configuration files and place them in specific location
    # Should create separate directory, Docker  "context" for each bgpd.conf file
    def generateConfigsAndReturnContext(self):

    # Given list of config files, make context.
    def __makeContext(self, bgpd_confs):

    # Given list of neighbor-listing portions that contain topology info,
    # builds 4 full bgpd.conf files (as lists of lines) , returns list of those lists
    def __makeConfigs(self, list_of_portions):


    # Build bgpd.conf file from template for specific router number, given total
number
    # For "Full Mesh" configuration
    @staticmethod
    def buildTopologyPortionMesh(num, curr, subnet):

    # Build bgpd.conf file from template for specific router number, given total
number
    # For "Hub" configuration
    @staticmethod
    def buildTopologyPortionHub(num, curr, subnet, hub_num):

    # Add timestamp to any name
    @staticmethod
    def addTimeStamp(name):

    # Return simple "neighbor IP" string for given router number
    @staticmethod
    def neighborString(subnet, no, remote_as):

    # Return X random prefixes from subnets.txt pool
```

```
    @staticmethod
    def getSubnets(num):


class ChaosLemurContextGenerator:

    def __init__(self, configs, root):
        se

def buildContext(self)

 def copyInFiles(self, bgpd_conf, no):

  # Build and run all generated containers
  def buildContainers(self):
```

class ChaosLemur:
Bold/Highlighted = public-facing
The ChaosLemur class is used for actually causing the failures and reversing them. It works assuming that the *only* docker containers running are ChaosLemur routers. It is thus advisable to clean the docker environment before starting any ChaosLemur experiments, using the src/CLGen/**clean.sh** script

```
class ChaosLemur:

    def __init__(self):

    # Take down Node
    def takeDownNode(self, num):

    # Take down link
    def takeDownLink(self, rt1, rt2):

    # Reverse all failures
    def reverseFailures(self):

    # Take down RANDOM node. Return ID of node failed.
    def takeDownRandomNode(self):

    # Take down RANDOM link. Return ID of link failed.
    def takeDownRandomLink(self):

    # Display prefixes loaded for router X
    def showIPRoute(self, rt_num):

    # Display prefixes loaded for ANY router that is still running, given dead one
    def showAliveIPRoute(self, dead_one):
```