PyNet2 Developer Guide

Author: Fazal Rahman, Hossein Amirinia

**Table of Contents**

[1. The Basics](#30j0zll)

[1.1. Installing Dependencies](#3znysh7)

[1.2. Notes on SimPy](#tyjcwt)

[2. Components of PyNet](#1t3h5sf)

[2.0.1. Limitations of this document](#2s8eyo1)

[2.1. Driver Function](#3rdcrjn)

[2.2. network.py](#lnxbz9)

[2.3. ieee802154 or 'Net'](#1ksv4uv)

[2.4. config.py](#2jxsxqh)

[2.5. Node.py](#3j2qqm3)

[2.5.1. The 'Node' class](#4i7ojhp)

[2.6. Clustering in PyNet](#1ci93xb)

[2.6.1. cluster.py](#2bn6wsx)

[2.6.2. Different Clustering Algorithms](#3as4poj)

[2.7. Messaging in PyNet](#23ckvvd)

[2.8. Other Important Modules](#32hioqz)

[2.8.1. energymodel](#41mghml)

[2.8.2. Interference](#vx1227)

[2.8.3. superframe](#1v1yuxt)

[3. Architecture of PyNet](#2u6wntf)

[3.1. Physical Layer](#3tbugp1)

[3.2. MAC Layer](#nmf14n)

[3.3. Application Layer](#1mrcu09)

[3.4. User Interface](#2lwamvv)

# Abstract

This document describes the various components of PyNet2 in detail. It is meant to facilitate the development and revisioning of PyNet2. It is meant as a glance into how the various components of the code interact. A detailed description of the API can be found in the API reference chart. This manual does cover the installation step but more detail about this step can be found in the user manual.

# The Basics

PyNet is a network simulator for WSN written in python3. It has an implementation of the IEEE 802.15.4 standard and is geared toward the research domain.

## Installing Dependencies

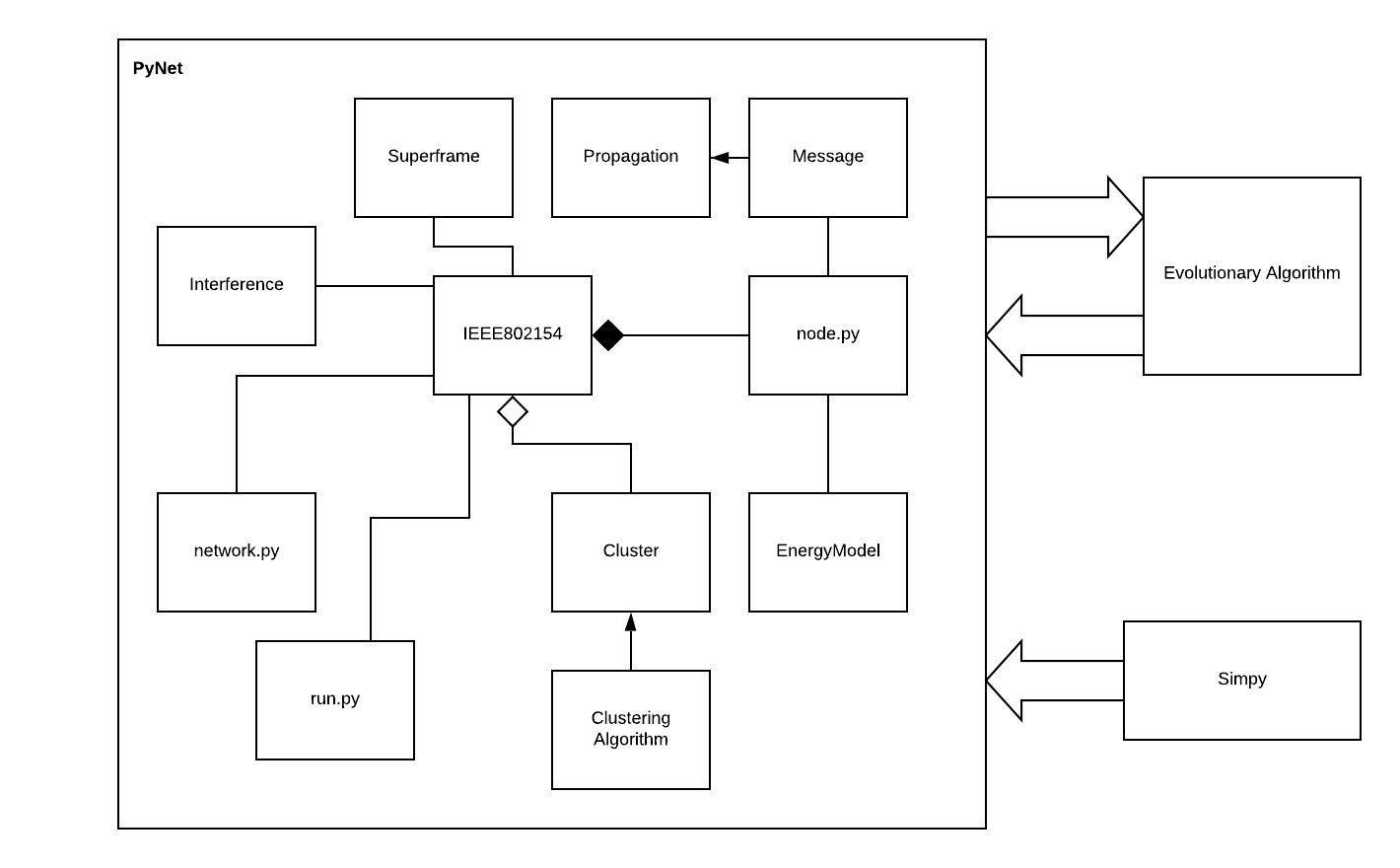
To install and run PyNet, first install python3. Use pip to install simpy, pandas, matplotlib etc. eg: pip install simpy

## Notes on SimPy

SimPy is a discrete event simulator written in python that takes advantage of generator functions. If you are unfamiliar with this concept, please visit the SimPy documentation located at <https://simpy.readthedocs.io/en/latest/>

# Components of PyNet

The following illustration is meant to be a top level description of the functioning of PyNet.

[Component Diagram](http://filef073431c339bd08801236297734070.odt/finalComponent.jpeg) 

The various classes are explained briefly in the following section based on their various functions.

### Limitations of this document

As can be inferred from the illustration above, not all modules are thoroughly explained in this document as the code is still under constant revision by the author. However, some modules are not included deliberately, such as 'config.py' as numerous other modules contain 'config.py' as a dependency and including it in the illustration will only add clutter. Surely, a description of all non-included modules will be provided in this document.

## Driver Function

The driver class is the script that bootstraps the simulator. In the above diagram, 'run.py' acts as the driver function. This module borrows an instance of the 'ieee802154.py' class initially defined at 'network.py' as well as an instance of the class containing the desired clustering algorithm to be used. To alter the runtime of the simulation, simply change the arguments to the env.run() method.

## network.py

This script is used by the driver function as a dependency. The script creates an instance of the 'ieee802154.py' class and adds a certain number of nodes to said instance. Finally, it prompts the instance method 'ieee802154.nodedsicovery()' which boostraps the collection of nodes added to the network. To keep things simple, this instance will be named as the 'Net Instance' or simply 'Net'. This is simply a convenient alias which will be helpful to understand this documentation.

The format for adding nodes to any object netobj: ieee802154 is as follows:

netobj.addnode(<node instance>)

## ieee802154

This class is arguably the most significant one as it defines the behaviour of the entire network’s MAC protocol. It is the class which defines an instance of the SimPy working environment 'env' used throughout the entire simulator. Thus, this is the primary event loop. It simulates the clock which is used to perform scheduling.

The 'superframe' class defines the intervals which the simpy clock defined by 'Net' uses to set the CSMA, TDMA and inactive regions of the superframe. The program perpetually cycles between these three states, during which every other component performs specific tasks. For instance, different nodes behave differently during these states. This is the chief reason why nodes have a reference to the 'Net' to which they belong.

After the nodes have been added to 'Net', clustering must take place. A cluster in PyNet is defined by a star-network with the sink node being the cluster head. This formation is handled by the clustering algorithms implemented. There are two such algorithms defined by PyNet, namely K-Means clustering and LEACH-C. More documentation about these algorithms may be found in the following: <https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html> <http://pdos.csail.mit.edu/decouto/papers/heinzelman00.pdf>

More details on cluster formation is provided in the Clustering section of this document.

## config.py

This is the file that contains the configurations for the runtime environment. The parameters defined here are constants used throughout the simulator. The variables are self explanatory and may be altered as per requirement. For reasons of simplicity, this component is not included in the illustration as the macros defined here are used by various other components. For instance, the macros 'CSMAduration', 'TDMAduration' and 'InactiveDuration' are used by the 'superframe.py' class which in turn is associated with 'Net'. Also, the macro 'BEACONING TIME' is used by 'node.py' for defining the behaviour of the base station. Let us explore this 'node.py' class.

## Node.py

This class defines a wireless sensor within PyNet. The enumeration SensorType defines two types of sensors, namely "Alert-Temperature" and "Monitoring". This can be easily extended to encompass other sensor types as well. The 'sensor' class defines methods that return random values that act as the primary data within the simulator.

### The 'Node' class

The objects instantiated from this class, namely, the nodes defined in 'network.py' keep a reference to the specific 'Net' to which they belong. Recall that we used the 'addnode()' method to add various nodes to the 'Net' in 'network.py'. An instance of EnergyModel is also kept by the 'Node' object referenced by the self.power attribute. Upon instantiation the run(self) method checks whether the particular node is the first in the list of all nodes in 'Net'. If it is, the node is assigned the role of the base station or simply 'BS'. The role of the 'BS' is quite significantly different from that of the regular nodes as it performs different operations based on the state of the 'Net.clock[0]' attribute. The 'EnergyModel' instance is used to model the energy drain of every node, with each one having a certain predefined power level. The class contains methods that deal with the nodes in the event of a depleted energy source.

## Clustering in PyNet

Clustering in PyNet takes place at the MAC layer, as defined by 'Net'. 'Net' holds reference to an array of objects of class 'cluster.py'. Each object in this array corresponds to a cluster within the network, with a maximum of seven nodes per cluster. All communications within clusters are scheduled to take place during the TDMA time slot of each source node belonging to the cluster.

### cluster.py

This class handles a collection of nodes. It contains class methods for adding and removing nodes to and from itself. Every cluster has a cluster head that is selected by the ClusterheadSelection() method which assigns a new cluster head based upon maximum remaining energy. There also exist methods to provide aggregate values for energy, light, and temperature received from nodes within the cluster. The formation of such Clusters is governed by the various clustering algorithms implemented in PyNet.

### Different Clustering Algorithms

The clustering algorithms are in charge of cluster formation and the determination of cluster heads.

1. LEACH-C

The LEACH-C method for clustering was the first algorithm implemented by PyNet. This algorithm is geared towards efficient energy consumption and cycles the role of cluster head among all the nodes in the cluster periodically. The nodes with the highest amount of energy remaining have higher chance of being cluster heads in the next iteration. The clusterformationarea method facilitates cluster formation based on the position of the node in the Cartesian plane. As of now, it defines nine clusters that together encompass the area of the plane. Furthermore, this method also sets the cluster heads for the individual clusters formed using methods from the 'cluster.py'

1. clusteringKMEANS

Kmeans clustering is a means to organize the clusters based on their distance from the nearest cluster head. The class that describes this is defined in the file named 'clusteringKMEANS.py'

## Messaging in PyNet

Messaging in PyNet is handled by the 'message.py' class. A shared instance of this component among all nodes and also the 'Net' suffices for communication. The clustering algorithms use this service to broadcast state changes.

The messaging model also uses a propagation model. Further, note that the “Energy Model” described in PyNet is only used in this component. That is to say, the energy loss is accounted for only during transmission and reception of messages.

## Other Important Modules

### energymodel.py

Defined in this class is a model of energy consumption for the wireless nodes. This model accounts for idle power loss as well as power lost during transmission and reception of packets. These parameters are set by default as identical as an assumption is made that transmission and reception costs the same energy. Improvements to this can be made by altering the decreasetxenergy and decreaserxenergy functions.

### Interference.py

The interference model for PyNet is described by this component. It has a method to check whether the Cartesian distance between any two nodes is less than a certain threshold, when they communicate in the same time slot. The threshold is the TXRANGE macro defined in the 'config.py' file.

### superframe.py

This class defines the intervals during which the active and inactive periods are defined. Its real implementation lies within the 'ieee802154.py' file.

### propagation.py

There are several propagation models used in PyNet. These models are used to predict the received signal power of each packet. At the physical layer of each wireless node, there is a receiving threshold. When a packet is received, if its signal power is below the receiving threshold, it is marked as an error and dropped by the MAC layer.

The various propagation models used are as follows:

1. Free Space
2. Two Ray Ground
3. Shadowing

More information about these models may be found in the following links:

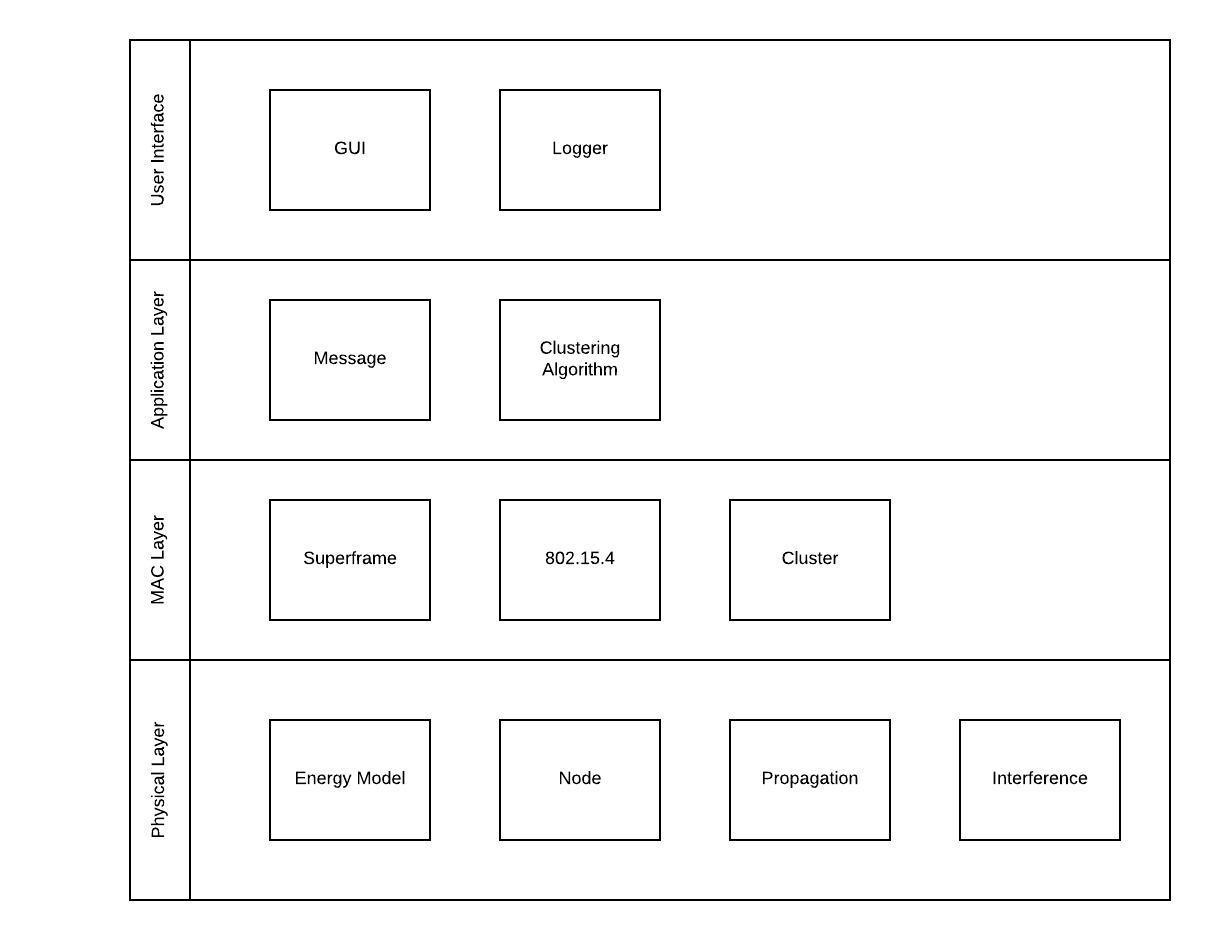
<https://www.wikiwand.com/en/Free-space_path_loss>

<https://www.wikiwand.com/en/Two-ray_ground-reflection_model>

<http://www.wirelesscommunication.nl/reference/chaptr03/shadow/shadow.htm>

# Architecture of PyNet

Since the object of the simulation is a network, it is best to adapt the different components of the simulator according to the OSI model of networking theory. The diagram below classifies the different modules of PyNet based on how they fulfill the requirements of each layer of the OSI model.



[Architecture of PyNet](http://filef073431c339bd08801236297734070.odt/overallArch.jpeg)

## Physical Layer

The physical layer of the OSI describes the specifications of the physical space and node hardware. In PyNet, the shared media that corresponds to the 3-dimensional space shared by the nodes is not modelled. Instead, the messages which are transmitted over this media are set to abide by a specific set of rules that mimic the behaviour of the media. To achieve this, the 'node' class provides detailed instructions for every instance of itself to carry out depending on whether the node is a cluster head or not. These set of instructions mimic the instruction set that may be found in the hardware of different motes.

The different motes must then have a concept of energy which they expend to perform their various functions. This energy model is also modelled within PyNet.

The classes 'propagation' and 'interference' model the characteristics of the wireless media.

## MAC Layer

Once the physical layer is set up to send and receive transmissions, it becomes imperative to describe the behaviour and structure of transmissions. This is handled by the MAC layer in PyNet. The behaviour of clusters and the 802.15.4 standard are implemented in this space.

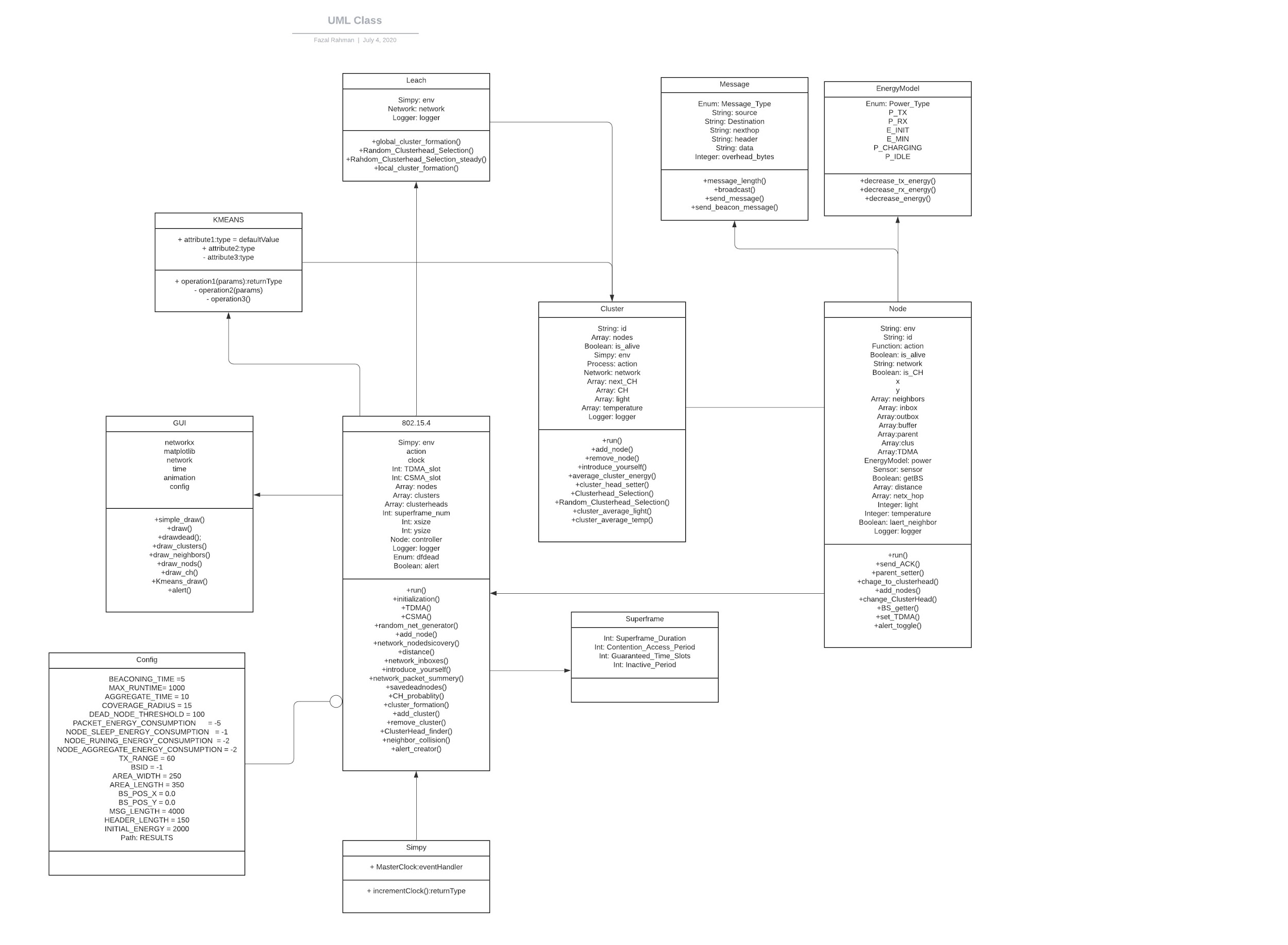
## Application Layer

To maintain the extensibility of PyNet, the 'message' and 'clustering formation' components are placed in the application layer. This ensures that the user of PyNet may be able to adapt PyNet to simulate various configurations of sensor networks.

## User Interface

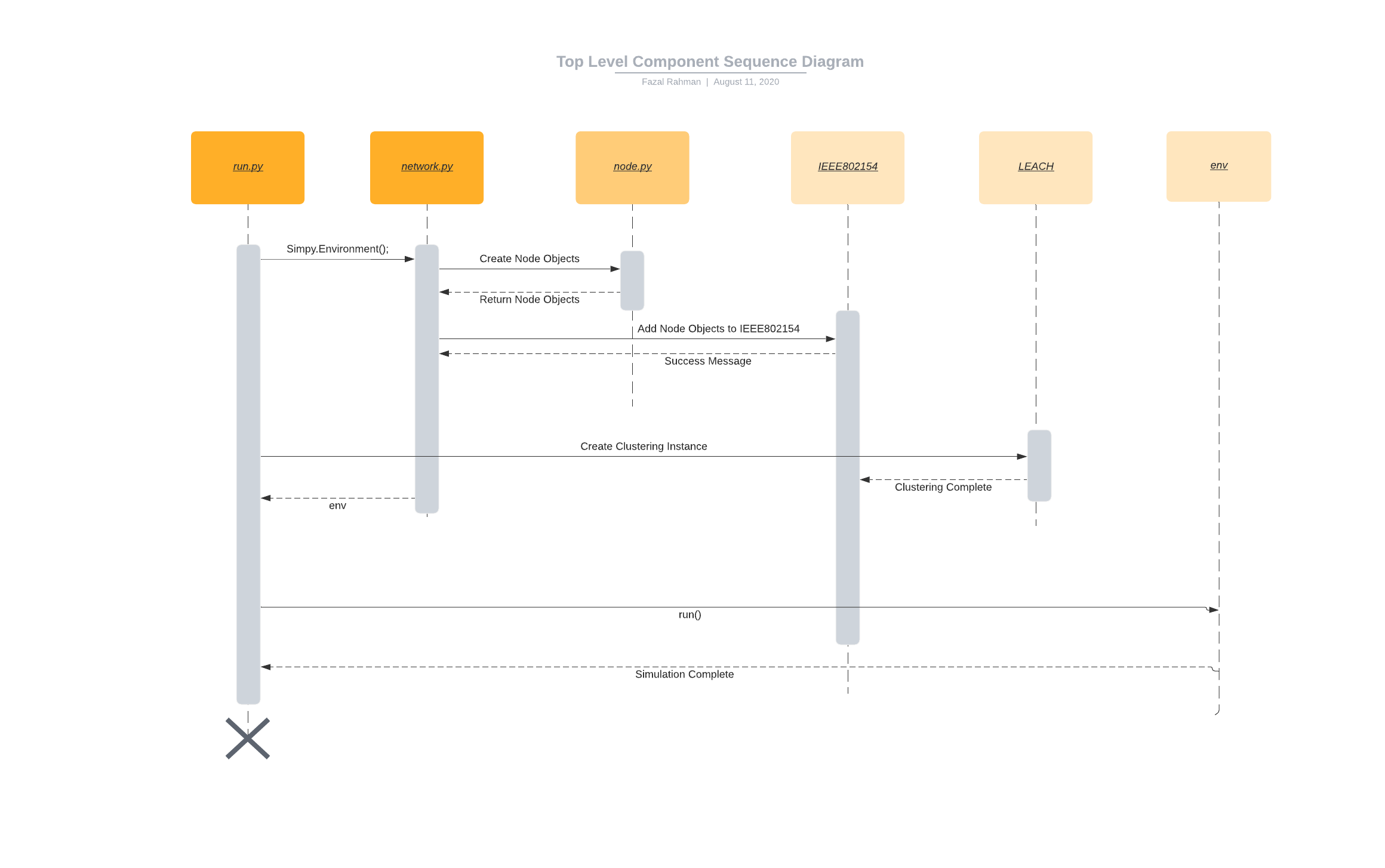
This is the top level component that aids in the visualization of the simulation as well as the logging of events yielded by the simulator.

# Class Diagram of PyNet2

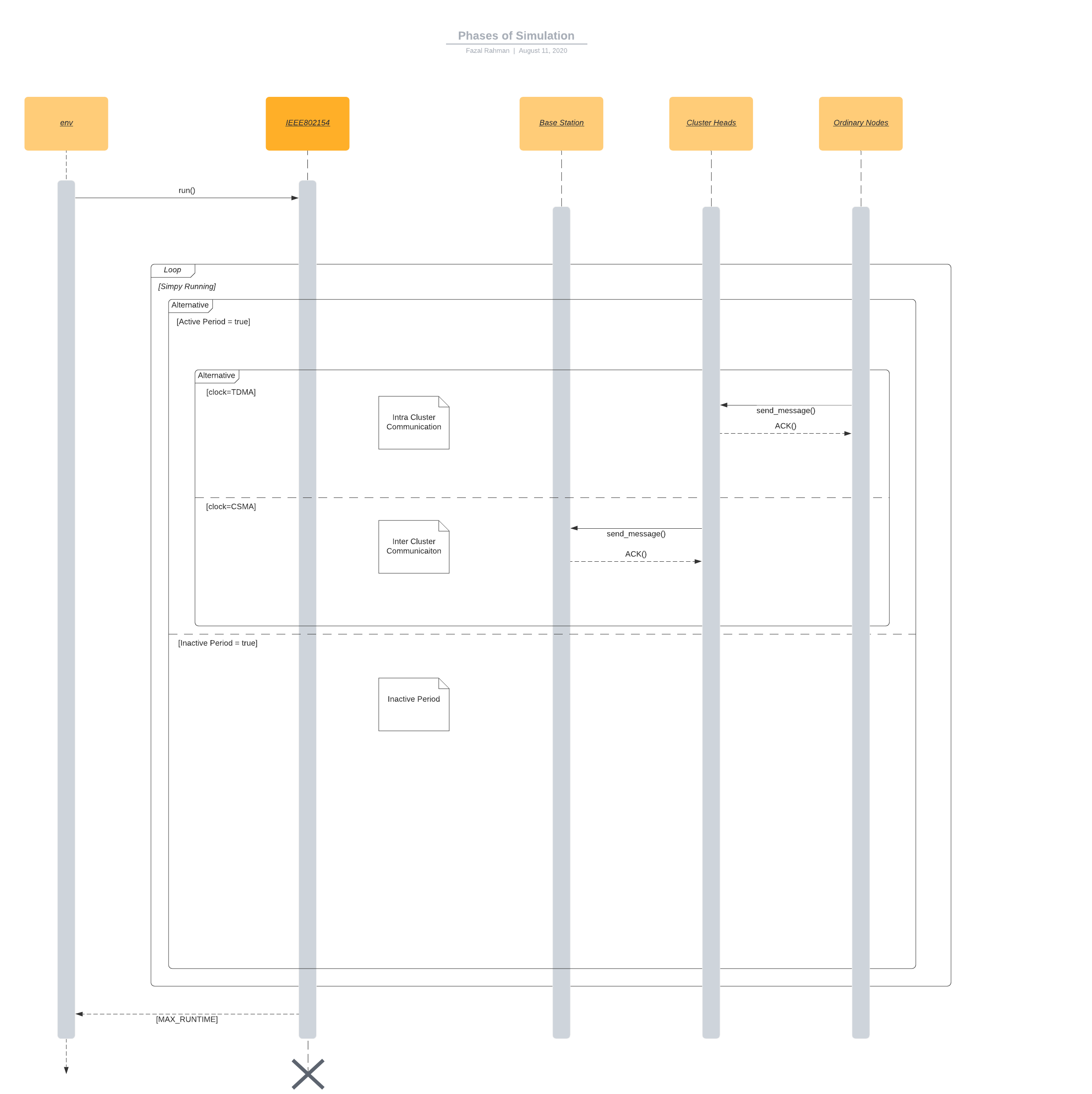


The above figure shows important associations between the classes present in PyNet2.

# Top Level Sequence Diagram



As depicted in the illustration above, *run.py* drives the simulation by initializing a Simpy.Environment() instance which is referenced by *env*. Once *env* is created, *network.py* creates instances of *node.py*, with the first being the base station. These nodes are then assigned to an instance of the “*802.15.4”* instance The role of the base station and other nodes within the network are depicted in the illustration below. These distinct roles for the nodes other than the base station are assigned by *LEACH.py.* After the clustering formation is complete, a *run()* command is issued to *env.*



During the three phases, different functionality is achieved. These functions may be altered by editing the “node.py” file.

The global clock that is produced by *env* is divided into three phases defined by *802.15.4* namedly the TDMA phase, CSMA phase and the inactive period. The inner workings during each of these phases are described by the communications of the nodes during said phases.

# 