California State Polytechnic University, Pomona

Electrical and Computer Engineering Department

ECE 4830 – Project Design Principles and Applications

Autonomous Self-Healing and Beamforming in 5G Networks

Connor Carr, Omar Naffaa, Cameron Krischel, Jonathan Chi, JJ Aguana, and Athan Alcala

Dr. Tamer Omar & Dr. Thomas Ketseoglou

May, 2020

Abstract

In order to meet increasing data demands and faster download/upload speeds required due to the transition from a 4G to a 5G network, we will have to upgrade our current physical communication infrastructure with new 5G technologies that can transmit at a minimum 1-10 Gigabits/second with less than 1 millisecond of latency. In order to meet these criteria, these networks will utilize new technologies including MIMO beamforming antennae, Millimeter Wavelengths (mmWave), and new network management techniques to increase data rates and network reliability while significantly reducing wasted power. Most notably, it will be important that the network is able to recover autonomously from network failure or congestion through a process called Self-Healing. Self-Healing processes are critical for these future networks because they are often likely to be comprised of a dense number of smaller cells; whereas older networks have cell service stations more spread out and spaced apart. With the greatly increased number of cells in a smaller service area, servicing base stations that are experiencing network failure in person or even remotely becomes impractical, and thus creates the need for the network to recover from failure on its own. Our project aims to simulate a network comprised of these new technologies in order to test various self-healing and network management strategies in the hopes of implementing a digital network manager capable of detecting the "unhealthy/failing" cells present in the network and implement these newfound self-healing procedures. The simulator will accurately model channel conditions and model downlink communications between the Base Stations (BS) and User Equipments (UEs). The project centers around determining the optimal algorithms implemented at the management level for achieving this result using the constraints introduced by the environment and technologies simulated.

Table of Contents

Introduction

Every day our demand for data grows, and new internet reliant technologies are being introduced all the time. It is expected within the next few decades that the Internet of Things will grow, self-driving cars will be communicating with each other, and people will be streaming videos in 4k on the go. In order to meet these demands, we will have to upgrade our current physical infrastructure with new 5G technologies. A 5G network will be required to provide between 1-10 Gigabits/second with less than 1 millisecond of latency. These networks will be using a variety of new technologies including: MIMO beamforming antennae, which will significantly reduce wasted power and allow for more reliable communication; Millimeter Wavelengths(mmWave), an unused portion of the EM spectrum that will allow for higher data rates; and new network management techniques to ensure network reliability. Notably, it will be important that the network be able to recover autonomously from network failure through a process called Self-Healing. Self-Healing processes are critical for these future networks because they will likely often be comprised with a dense amount of smaller cells; whereas older networks had base stations more spread out and spaced apart. 5G coverage will require many small cells to maintain coverage in dense environments. With the greatly increased number of cells, dealing with network failure in person or even remotely becomes impractical, and thus it will be ideal for the network to recover from failure on its own and maintain quality of service for the users. Our project aims to be capable of simulating a network comprised of these new technologies in order to test various self-healing and network management strategies in the hopes of implementing a digital network manager.

MIMO, multiple-input and multiple-output is an antenna technology that was introduced into many wireless communications systems including 4G LTE to improve signal performance [1]. Multiple antennas are both used as a transmitter and receiver. Using the processing power available in the communications circuit, the transmitter and receiver are combined to provide improvements in data rate and signal to noise ratios. The multiple antennas on the receiver and transmitter utilize the multi-path effects that always exists to transmit additional data, rather than cause interference. Through the use of multiple antennas, MIMO is able to utilize the multiple path propagation that exists to provide improvements on signal performance. While MIMO adds complexity to the system of wireless technology, it is able to provide significant improvements in performance. MIMO is currently being used in many high data rate technologies including Wi-Fi and other wireless and cellular technologies, such as 4G. Massive MIMO achieves everything a traditional MIMO with better results.

Massive MIMO is an extension of MIMO with more antennas. Traditional MIMO relies on four antennas, where massive MIMO features dozens that rely on signal processing to find the best and fastest way to route data to their destinations. With more antennas to utilize, massive MIMO provides faster and better spectral efficiency. In an experimental result provided in [2] the group achieved an increase in rate of efficiency that was 22 times better than existing 4G networks. Massive MIMO also faces some challenges in improvements. Since massive MIMO uses dozens of antennas, their components would cost more. The need for making many low-cost low-precision components that would effectively work together would drastically improve massive MIMO. In addition, as more and more antennas are being utilized, power consumption rises.

Reducing internal power consumption is a must to achieve an effective total energy efficiency. While there are numerous challenges that massive MIMO encounters, its benefits outweigh its costs.

5G networks are being designed to provide users bandwidth in the tens of gigabits/second; in order to achieve this, higher frequency/shorter wavelength solutions are required. The mmWave band extends from 30GHz to 300Ghz and offer numerous advantages over existing technologies [3]. Using these higher frequencies allow not only for higher bandwidth, but also have desirable properties including; antennae can be placed directly on a chip due to a smaller size, power loss reduction, and the range limitations which actually lend themselves well to frequency reuse. mmWave lends itself well to use with beamforming MIMO antennae, which results in narrow beams at high frequencies that have very low attenuation losses and thus require less power [4].

However, to properly model a network using these parameters, several channel conditions need to be considered and calculated for a realistic result. Perhaps the most important, SNR(Signal-to-Noise-Ratio) needs to be calculated for a given channel in order to determine the data rates that can be provided to the user, and the power consumption of the transmitter, along with the received power. It is also important to have a model for determining the path loss exponent for a given environment, because that will also impact the channel's data rates and power consumption as it is important in calculating the SNR [5]. It can be useful to examine the Consumption Factor (CF), which describes the maximum data rates that can be achieved for a given power consumption as a function of the channel conditions. It is desirable to maximize the ratio between the data rates achievable by the system and the power consumed by the system [6].

There are a great number of variables to consider in accurately modeling the physical nature of modern wireless networks. Modeling difficulties due to the inherently high scattering of mmWave, and the dependence on the specific environment on the performance of the system [7]. Due to this, our simulator will need to maintain a certain level of flexibility so that multiple methods of channel generation can be used for different simulation profiles.

A self-healing, self-optimizing network is organized throughout a layered management system. Self-healing is going to focus on the radio access network (RAN) layer. This is the layer that includes the base stations, which need the capability for transmitting conditional information for any self-healing functions required in the process. Base stations communicate with a controller-level element manager (EM), which collects necessary information to provide self-healing.

Centralized control happens at the core network (CN) layer. This layer manages the system at a high level by overseeing processes for centralized tasks, including monitoring base station conditions. The CN layer includes network manager (NM) controllers which communicate with lower level domain manager (DM) controllers from the RAN layer. These two controllers have similar functions, but they differ in the management level that they operate in.

The general 5G network will be based on small cells. A relevant approach entails using a phased array of antennas at the macro-cell base station (MBS), where it will be able to achieve massive MIMO beamforming [8]. Implementations of this network should preferably be open to the important aspect of programmability. By doing so, it would allow for ease of management by having an adaptable system where customizations and changes could easily be applied to certain functions. This provides a way to design a system with increasing performance efficiency. For example, different SNR calculations or scheduling and beamforming algorithms could be tested and implemented for optimization [9]. Further, the higher level controllers would maintain global control of the network, which provides management optimization regarding resource allocation and base station scheduling [10].

The transition from 4G to 5G will include a migration from eNodeB to small cells, which would be placed much more densely in a given area and result in a significantly higher amount of total towers to manage. As mentioned in [11] approximately 23% to 26% of revenue acquired from mobile cellular network revenue is spent on operation costs associated with the network, which is a cost that will increase due to the additional infrastructure that would have to be implemented in a 5G system. The solution to this issue is to implement a Self-Organized Network (SON) that would automate the network management process. A Self Organized Network would break down into 3 high level components: self-configuration, self-optimization, and self-healing.

Based on the research conducted in [12] self-healing can be classified into 3 main phases: detection, diagnosis and compensation. From these parameters a proposed self-healing model can be implemented as 5 modules that would be implemented in a way that would meet 3 objectives defined. Our model for the self-healing process would include fault monitoring, fault detection, fault diagnosis, system compensation, and system recovery. Fault monitoring functions include performing self-healing monitoring functions as suggested by [13]. This component of the self-healing model would also perform comparisons with our operational measurements in fixed time intervals, and if a fault is detected it is the responsibility of this portion of the model to generate a fault detection alarm. The second component of our model, fault detection, will take signals sent from the fault monitoring portion and perform self-healing diagnosis functions outlined in [13]. It will notify system operations and deliver critical information needed to declare a fault occurrence, determine the nature of the fault and specify if it should be Automatically Detected and Automatically Cleared (ADAC) or Automatically Detected and Manually Cleared (ADMC), and trigger diagnosis procedures if it is determined that a fault has occurred.

Before performing system compensation, the fault will be diagnosed in the third part of the model. In this phase more information is gathered to aid in identifying a probable cause of the detected fault, determine the main cause of this fault, and collect data and conduct some form of analysis to predict future faults more accurately. Once this is complete a severity level will be assigned to the detected fault and a signal is sent that would initialize a recovery procedure and possibly a compensation procedure if the fault is severe.

At this stage in the model the system will attempt to recover itself and provide compensation if necessary. System compensation includes surveying the ability of neighboring cells to share unused available resources, and then use the unallocated resources to compensate for the faulty cell until a full recovery is made. Once compensation is complete the digital system manager will return the resources back to their respective small cell. System recovery entails performing available tasks that can be used to repair the system, and if these automated procedures do not resolve the issue system recovery is responsible for initiating an ADMC to get the resources needed to fix the problem manually. This should resolve the issue, and as a result the alarms will be cleared, normal operation will resume, and a fault report will be generated with information pertinent to future system diagnosis and improvement.

System Architecture for 5G Network Simulator

The system is composed of 3 stages, which are (1) Initialization, (2) Simulation, and (3) Output. The initialization stage was implemented in the form of a Graphical User Interface (GUI), which allows the user to create a network environment of eNodeBs represented by interconnected hexagons by clicking to build a structure for testing. Once this portion of the initialization is complete, the user is directed to the next window and can begin specifying parameters to be used in the simulation. Some examples of this include the number of antennae within an eNodeB and the number of transceivers per antenna. After setting the parameters the simulation stage can begin.

In the simulation stage the eNodeB structure is formed with appropriate network conditions based on parameters passed from the GUI as well as those taken from a CSV sheet containing accurate data rates and signal to noise (SNR) ratios simulated using a MATLAB script [see section: data rates and power calculations]. The environment is ran for a specified amount of time, allowing us to both analyze network conditions in the program in real time while also producing output CSV files for data analytics once the simulation terminates.

The output of the simulation will be analyzed in real-time using data captured for a specified amount of time within the system (e.g. the most recent 10 seconds) so we can determine the condition of our network simulation at any given time without having to halt the simulation. Details on this process will be defined later in our methodology.

Shown in figure 1 is the block diagram that summarizes the process above at a high level:

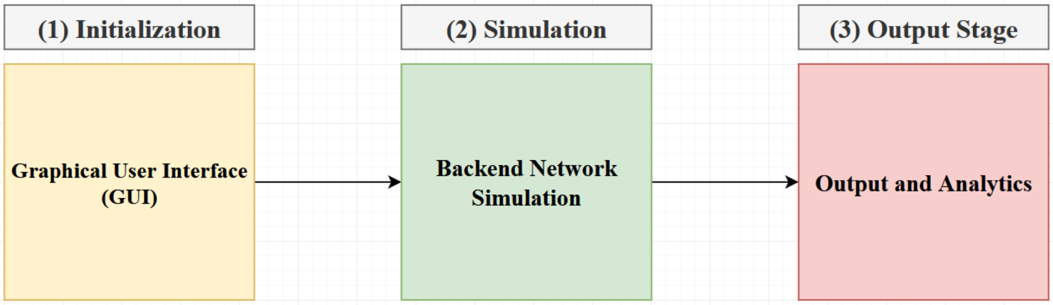


Figure 1: System Architecture

The simulator we designed as a part of this project is meant to depict various real-world scenarios of systems using 5G technology. The simulator starts off with an initial setup screen where the user generates hexagons on a grid depicting the various Base Station nodes. Then, a second screen allows the user to set values for duration of the simulation, how many simulations are to be run, and more.

Shown in figure 2 is a process diagram showing the stages of the Graphical User Interface:

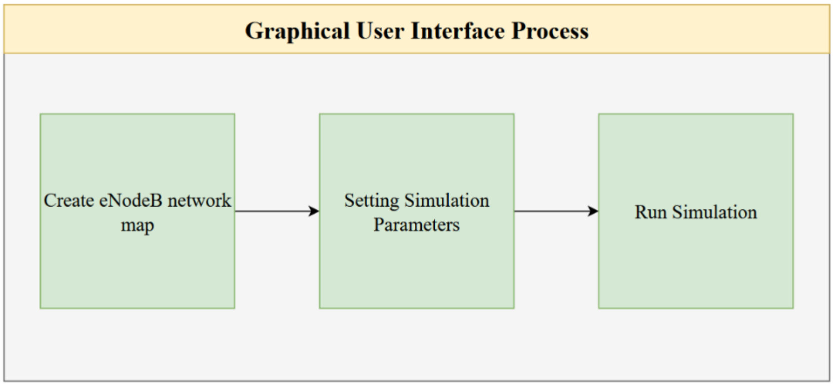
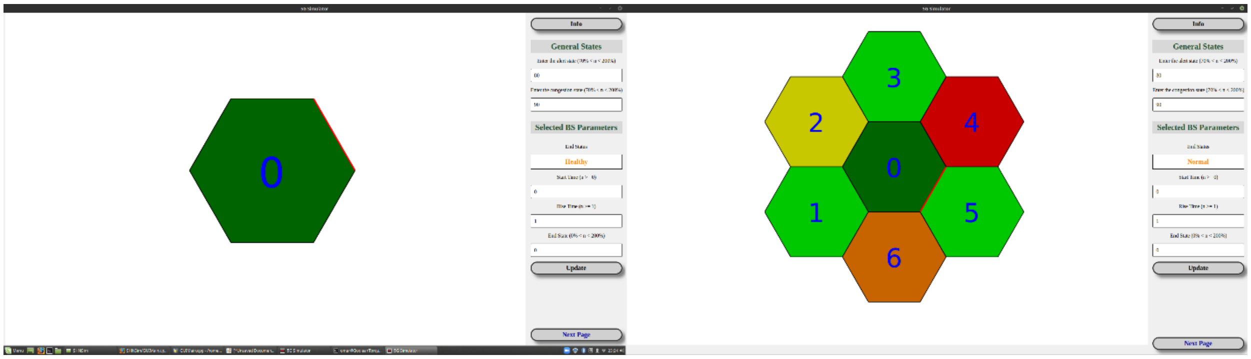

Graphical User Interface Process
Create eNodeB network map
Setting Simulation Parameters
Run Simulation

Figure 2: Graphical User Interface Block Diagram

The sections below elaborate on the stages of the GUI in detail.

<u>Creating a network map:</u>

In the first stage of the GUI, the user is presented with a window with a single hexagon in the center, representing the first eNodeB. The white numbers represent the station number. Additional hexagons (eNodeBs) are placed by left clicking. The edge that is closest to the mouse location will highlight in red, representing where the new hexagon will be placed. New hexagons can only be placed adjacent to the selected hexagon. To change which hexagon is selected, simply left click on a new hexagon. The selected hexagon will be dark green, while all others will be light green. The first stage of the GUI is shown in figure 3, with 2 configurations: the initial screen the user is greeted with along with an arbitrary configuration



Enter the alert state (70% < n < 200%)
80
Enter the congestion state (70% < n < 200%)
90
End Status
Healthy
Start Time (n > 0)
0
Rise Time (n >= 1)
1
End State (0% < n < 200%)
0

Enter the alert state (70% < n < 200%)
80
Enter the congestion state (70% < n < 200%)
90
End Status
Normal
Start Time (n > 0)
0
Rise Time (n >= 1)
1
End State (0% < n < 200%)
0

Figure 3: First Stage Initial Screen vs Arbitrary Configuration

Each hexagon has a state, represented by a different color. To cycle through the 4 states, left click on the selected hexagon. The states and their corresponding colors are listed below.

Green = Healthy

Yellow = Congested from users

Orange = Congested from demand

Red = Failing

The sample configuration in figure 3 shows a scenario with eNodeBs that have different conditions (denoted by the color of the base station). We can see in this example the center eNodeB is healthy since it is green along with eNodeBs 2, 4, and 6. Additionally, eNodeB 1 is congested from the amount of users on the network, eNodeB 3 is congested by the amount of data being demanded from the user, and eNodeB 5 is failing.

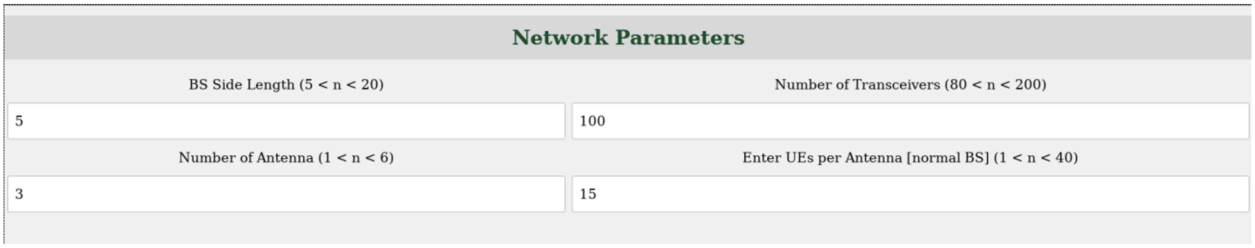
There are additional parameters on the right side of the first window of the GUI: "General States" and "Selected BS Parameters". Parameters under "General States" allows the user to set an alert level that can be used to determine when an eNodeB is approaching congestion and a congestion level that signifies when the eNodeB is actually congested. Parameters under "Selected BS Parameters" are configured per eNodeB, and are used to configure when a specific eNodeB will reach the condition it is set to. As an example, if we modified eNodeB 1 to be in congestion we would set the condition on the GUI and specify the parameters on the right so it will "ramp up" to the selected condition as opposed to jumping up to that level of congestion immediately.

Hexagons can be deleted by right clicking but cannot be arranged in a way that would create floating islands. Any number of hexagons can be instantiated, and the hexagons scale and center when additional hexagons are placed. On the top right corner of this stage of the GUI there is a button the operator can click called "Info", which provides basic information about operating the GUI such as the color code and the process of instantiating additional eNodeBs (e.g. left click an edge to add, right click to delete, etc.). After the user has created a setup, they can proceed to the next stage. The locations, states, and numbers of the stations are passed to the second stage in order to create a final "bundle" of information that will be used to create and run the simulation environment.

After instantiating the eNodeB structure in the previous stage of the simulation setup process the operator would then need to define initial parameters for the simulation to run. These parameters are the grouped into 3 categories:

1. Network Parameters
2. Simulation Parameters
3. Self-healing Parameters

<u>Network Parameters:</u>



Network Parameters

BS Side Length (5 < n < 20)
5

Number of Transceivers (80 < n < 200)
100

Number of Antenna (1 < n < 6)
3

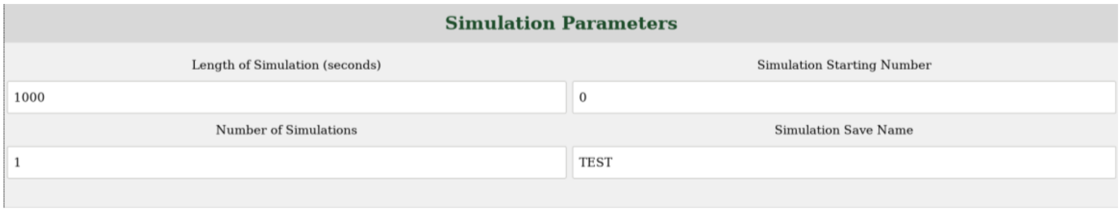
Enter UEs per Antenna [normal BS] (1 < n < 40)
15

Figure 4: Network Parameter Configurations

Shown in figure 4 is the first set of parameters needed in to initialize a simulation. The parameters above are the side length of the eNodeBs, the number of antennas per eNodeB, the number of transceivers on an antenna, and the number of UEs per antenna. The parameters, as noted on the Graphical User Interface, must be specified within a specific range in order to be valid.

The parameters above pertain to the network environment setup since all the items specified are models of physical components of any network. Items specified here are virtually created in the system to be a physical thing (e.g. an eNodeB is a cellular tower while a UE could be a computer or cell phone).

Simulation Parameters:



Simulation Parameters

Length of Simulation (seconds)
1000

Simulation Starting Number
0

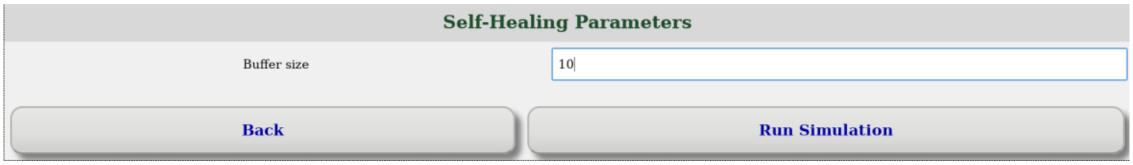
Number of Simulations
1

Simulation Save Name
TEST

Figure 5: Simulation Configuration Parameters

The next set of parameters that will be entered are the simulation parameters shown above. The length of simulations is in terms of a "tick", which is the fundamental unit of simulator time. A tick represents the time it takes for one set of instructions to be performed. For the purpose of our simulation specifically we equated the tick to a second. The next parameter is the number of simulations needed, which is specified should the operator want to run a batch of simulations without having to reinitialize the parameters multiple times. If there is a break in the set of simulations being ran and you have to restart the simulator, you could specify the simulator number that was last ran as the one you would like to start with. If you are running an entirely new set, you would have to enter "0" as the start number. Lastly, a save name for the simulator output has to be specified, otherwise a default name of "TEST" will be applied to all files outputted by the simulator.

Self-Healing Parameters:

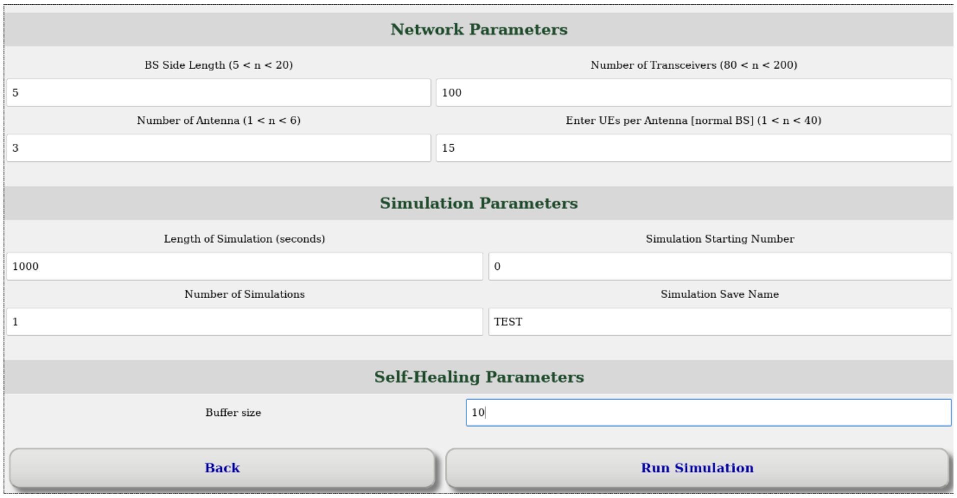


Self-Healing Parameters

Buffer size
10

Figure 6: Self-Healing Configuration

Finally, the self-healing parameter that would need to be specified for the network is the size of a buffer used in the backend of the system. The buffer structure built into the backend is used to capture data during the simulator runtime for analytics in order to perform our self-healing procedure. Once these parameters are entered, the simulation can begin running by pushing the button "Run Simulation". For reference, figure 7 (below) will be provided to give the user an idea of how each piece of the parameter entry stage looks like as opposed to the pieces provided above.



Network Parameters

BS Side Length (5 < n < 20): 5

Number of Transceivers (80 < n < 200): 100

Number of Antenna (1 < n < 6): 3

Enter UEs per Antenna [normal BS] (1 < n < 40): 15

Simulation Parameters

Length of Simulation (seconds): 1000

Simulation Starting Number: 0

Number of Simulations: 1

Simulation Save Name: TEST

Self-Healing Parameters

Buffer size: 10

Figure 7: Full Parameter Window View

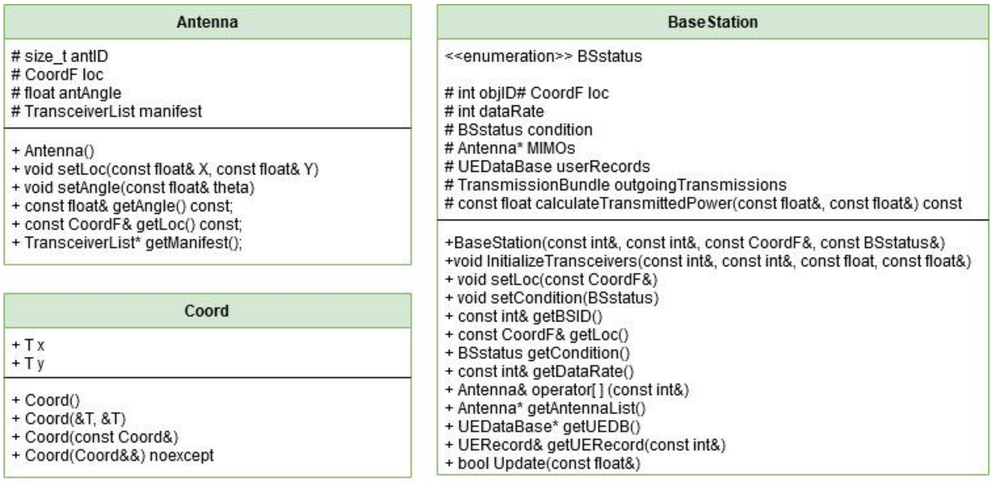
<u>Additional Details:</u>

The simulator parameters contain input validation to ensure that numerical values entered for the integer parameters are of type “int” while also validating that the values obtained are within range. Additionally, the name of the simulator is validated by making sure that the entered name of the simulator has no special characters and is not an empty string.

<u>Backend Overview:</u>

This project is largely software-based; therefore, the design will mostly include different portions of the UMLs written for the duration of the project. By describing key classes written the hope is to thoroughly document the backend design. Other sections will cover the math and other specific details within the program; this section mainly aims to cover the program at a high level. Due to the largeness of the UML, connection lines from one class to another are omitted.

Antenna, BaseStation, and Coord:



```
Antenna
# size_t antID
# CoordF loc
# float antAngle
# TransceiverList manifest

+ Antenna()
+ void setLoc(const float& X, const float& Y)
+ void setAngle(const float& theta)
+ const float& getAngle() const;
+ const CoordF& getLoc() const;
+ TransceiverList* getManifest();
```

```
Coord
+ T x
+ T y

+ Coord()
+ Coord(&T, &T)
+ Coord(const Coord&)
+ Coord(Coord&&) noexcept
```

```
BaseStation
<<enumeration>> BSstatus

# int objID# CoordF loc
# int dataRate
# BSstatus condition
# Antenna* MIMOs
# UEDataBase userRecords
# TransmissionBundle outgoingTransmissions
# const float calculateTransmittedPower(const float&, const float&) const

+BaseStation(const int&, const int&, const CoordF&, const BSstatus&)
+void InitializeTransceivers(const int&, const int&, const float, const float&)
+ void setLoc(const CoordF&)
+ void setCondition(BSstatus)
+ const int& getBSID()
+ const CoordF& getLoc()
+ BSstatus getCondition()
+ const int& getDataRate()
+ Antenna& operator[ ] (const int&)
+ Antenna* getAntennaList()
+ UEDataBase* getUEDB()
+ UERecord& getUERecord(const int&)
+ bool Update(const float&)
```

Figure 8: Class Diagrams 1

I. Coord:

   a. The coord class is implemented using as a template class, which allows the user to specify the coordinates needed (i.e. whether the accuracy of a floating-point number is needed or if a whole number (integer) location is going to be specified). This class is widely used across the program to represent a location on a virtual grid, and as such many other classes will include this class.

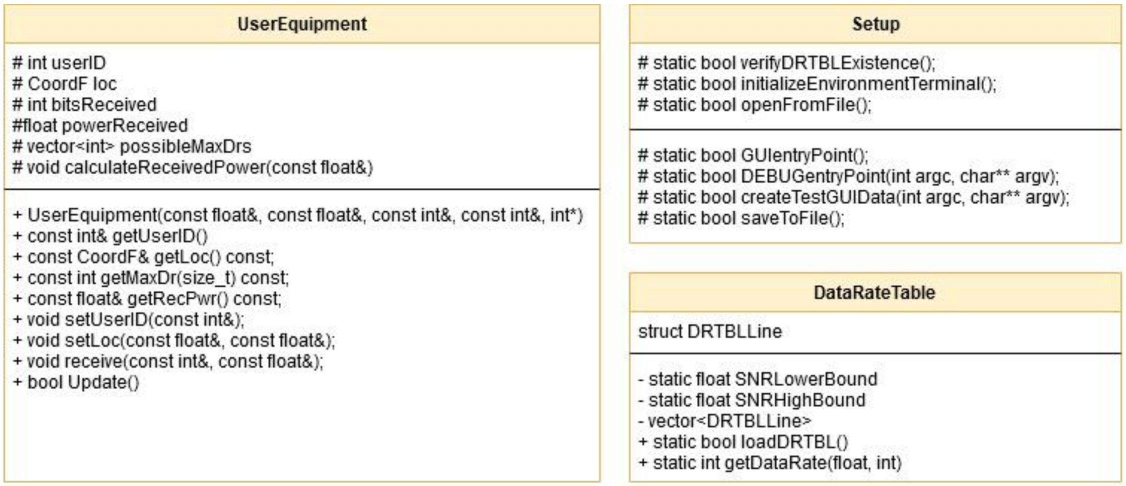
II. BaseStation:

   a. The BaseStation class implements the hexagons drawn with the GUI portion as base stations (eNodeBs) within the program. A base station, in theory, should contain a number of antennas that will be used to connect users to a base station as well as a list of user records to document the users connected within a particular eNodeB. The UML for BaseStation, as well as the program files, show all the available functions within the BaseStation class.

III. Antenna:

   a. The antenna class is the program's way of implementing antenna objects within the network simulation. The main components of an antenna are its location, angle and the transceiver list for a specific antenna object. The transceiver list contains a record of all transceivers on an antenna, and can be used for things such as determining users connected to an antenna.

UserEquipment, Setup, and DataRateTable:



```
UserEquipment
# int userID
# CoordF loc
# int bitsReceived
#float powerReceived
# vector<int> possibleMaxDrs
# void calculateReceivedPower(const float&)
+ UserEquipment(const float&, const float&, const int&, const int&, int*)
+ const int& getUserID()
+ const CoordF& getLoc() const;
+ const int getMaxDr(size_t) const;
+ const float& getRecPwr() const;
+ void setUserID(const int&);
+ void setLoc(const float&, const float&);
+ void receive(const int&, const float&);
+ bool Update()
```

```
Setup
# static bool verifyDRTBLExistence();
# static bool initializeEnvironmentTerminal();
# static bool openFromFile();
# static bool GUIentryPoint();
# static bool DEBUGentryPoint(int argc, char** argv);
# static bool createTestGUIData(int argc, char** argv);
# static bool saveToFile();
```

```
DataRateTable
struct DRTBLLine
- static float SNRLowerBound
- static float SNRHighBound
- vector<DRTBLLine>
+ static bool loadDRTBL()
+ static int getDataRate(float, int)
```

Figure 9: Class Diagrams 2

I. User Equipment

   a. User equipment objects populate the program since they serve as the user devices on a network (e.g. a cellular phone, computer, or other internet connect device). They are attached to a transceiver on an antenna, and the antenna itself is within an eNodeB. Sending data from an eNodeB to the user equipment uses a certain amount of power based on the Signal to Noise Ratio (SNR) provided by the data rate CSV provided by a supplemental MATLAB script (see data rate and power calculations section below).

II. Setup

   a. The setup function primarily serves to transfer the program from running the GUI, to running the simulation based on the entered parameters. The debug entry point for the GUI is used should the user want to debug on Windows since the GUI interface works only on Linux.

III. DataRateTable

   a. The data rate table class is used to load the data rate table that provides the SNR values into memory while also providing an interface to use it with. The data rate table values are used to give a user connected to the eNodeB a random SNR value between -12 db and 12 db in order to provide the network with a realistic model for random SNR value generation. The process the script takes to generate these SNR values is covered in more details in the section “Data Rate and Power Calculations”.

EnvironmentInitialization, FileIO, and IRPManager:

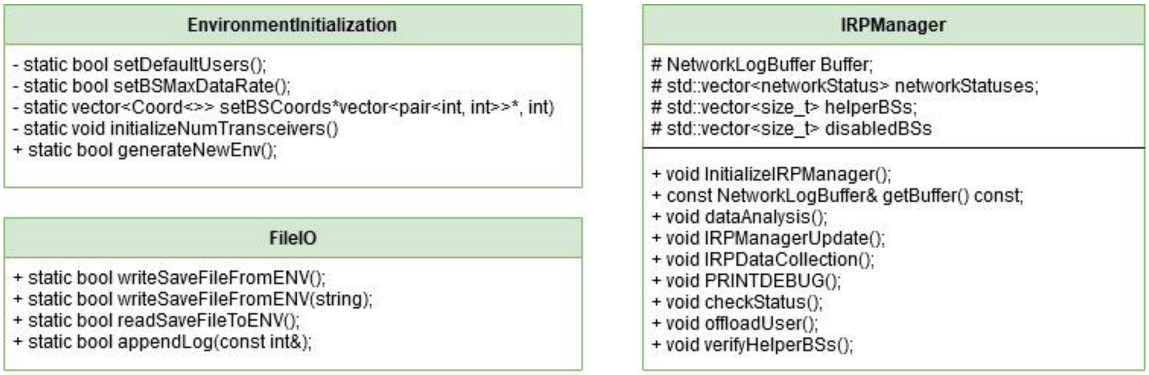

EnvironmentInitialization
- static bool setDefaultUsers();
- static bool setBSMaxDataRate();
- static vector<Coord<>> setBSCoords*vector<pair<int, int>>*, int)
- static void initializeNumTransceivers()
+ static bool generateNewEnv();
FileIO
+ static bool writeSaveFileFromENV();
+ static bool writeSaveFileFromENV(string);
+ static bool readSaveFileToENV();
+ static bool appendLog(const int&);
IRPManager
# NetworkLogBuffer Buffer;
# std::vector<networkStatus> networkStatuses;
# std::vector<size_t> helperBSs;
# std::vector<size_t> disabledBSs
+ void InitializeIRPManager();
+ const NetworkLogBuffer& getBuffer() const;
+ void dataAnalysis();
+ void IRPManagerUpdate();
+ void IRPDataCollection();
+ void PRINTDEBUG();
+ void checkStatus();
+ void offloadUser();
+ void verifyHelperBSs();

Figure 10: Class Diagrams 3

I. EnvironmentInitialization

   a. The purpose of this class is to initialize an environment by settings various parameters before generating a new environment. This includes setting the location of the base stations as well as their data rates and initializing the number of transceivers in the antenna.

II. FileIO

   a. FileIO is used primarily to write the simulator data to an output CSV file for post processing. For each “tick” that occurs during the runtime of the simulator a new row of data is appended to a CSV file that documents the system characteristics.

   b. Each appended row contains the following:

      i. Time
      ii. BaseStation Parameters (ID and coordinates)
      iii. Antenna Parameters (ID, sector it covers)
      iv. Transceiver Parameters (coordinates and angle of transceiver)
      v. User Equipment Parameters (ID, coordinates)
      vi. Data rates (max, demanded, real)
      vii. BaseStation SNR and UE received SNR

III. IRPManager

   a. The IRPManager class contains functions used for the self-healing algorithm as well as other network manager functions. One key function in this class is determining which eNodeBs are need self-healing, which eNodeBs need help from self-healing, and which eNodeB is the best one to offload a user to. This algorithm will be covered in a later section.

NetworkLogBuffer, Transceiver, Transmission:

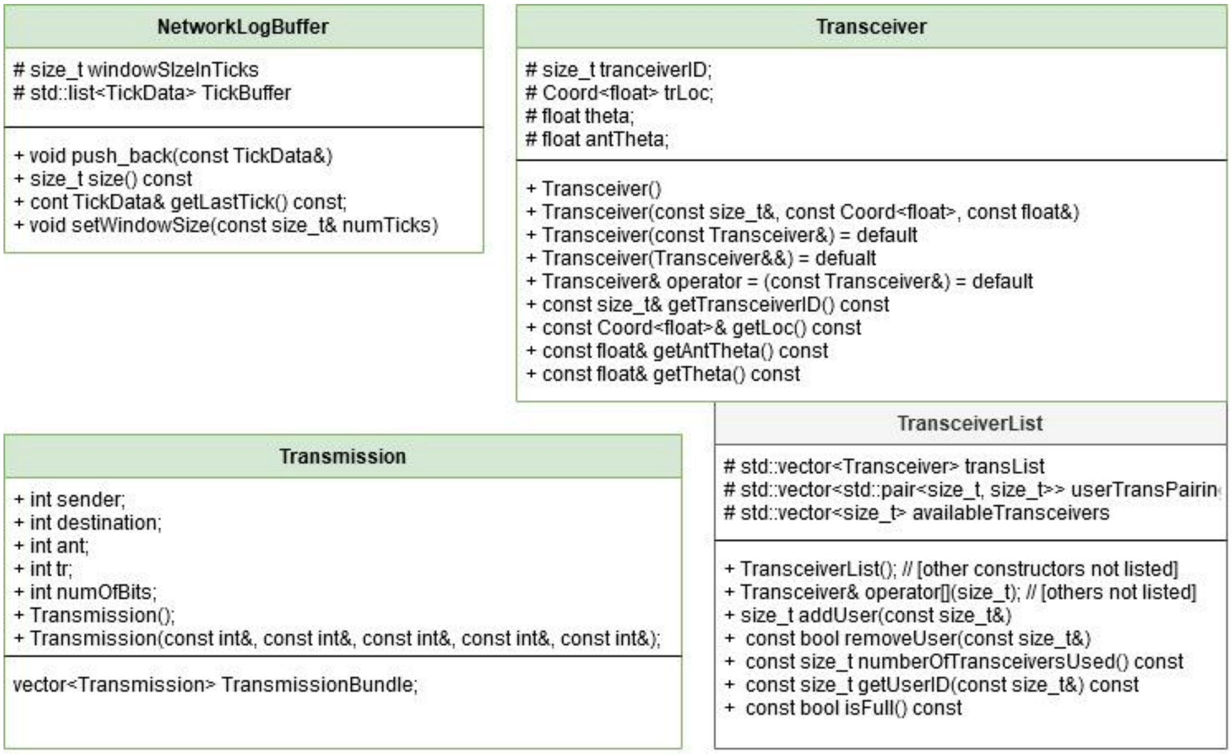

NetworkLogBuffer
# size_t windowSIzeInTicks
# std::list<TickData> TickBuffer
+ void push_back(const TickData&)
+ size_t size() const
+ cont TickData& getLastTick() const;
+ void setWindowSize(const size_t& numTicks)
Transceiver
# size_t tranceiverID;
# Coord<float> trLoc;
# float theta;
# float antTheta;
+ Transceiver()
+ Transceiver(const size_t&, const Coord<float>, const float&)
+ Transceiver(const Transceiver&) = default
+ Transceiver(Transceiver&&) = defualt
+ Transceiver& operator = (const Transceiver&) = default
+ const size_t& getTransceiverID() const
+ const Coord<float>& getLoc() const
+ const float& getAntTheta() const
+ const float& getTheta() const
Transmission
+ int sender;
+ int destination;
+ int ant;
+ int tr;
+ int numOfBits;
+ Transmission();
+ Transmission(const int&, const int&, const int&, const int&, const int&);
vector<Transmission> TransmissionBundle;
TransceiverList
# std::vector<Transceiver> transList
# std::vector<std::pair<size_t, size_t>> userTransPairin
# std::vector<size_t> availableTransceivers
+ TransceiverList(); // [other constructors not listed]
+ Transceiver& operator[](size_t); // [others not listed]
+ size_t addUser(const size_t&)
+ const bool removeUser(const size_t&)
+ const size_t numberOfTransceiversUsed() const
+ const size_t getUserID(const size_t&) const
+ const bool isFull() const

Figure 11: Class Diagrams 4

I. NetworkLogBuffer

   a. As the name implies, the network log buffer is used to hold data from the network.

II. Transceiver

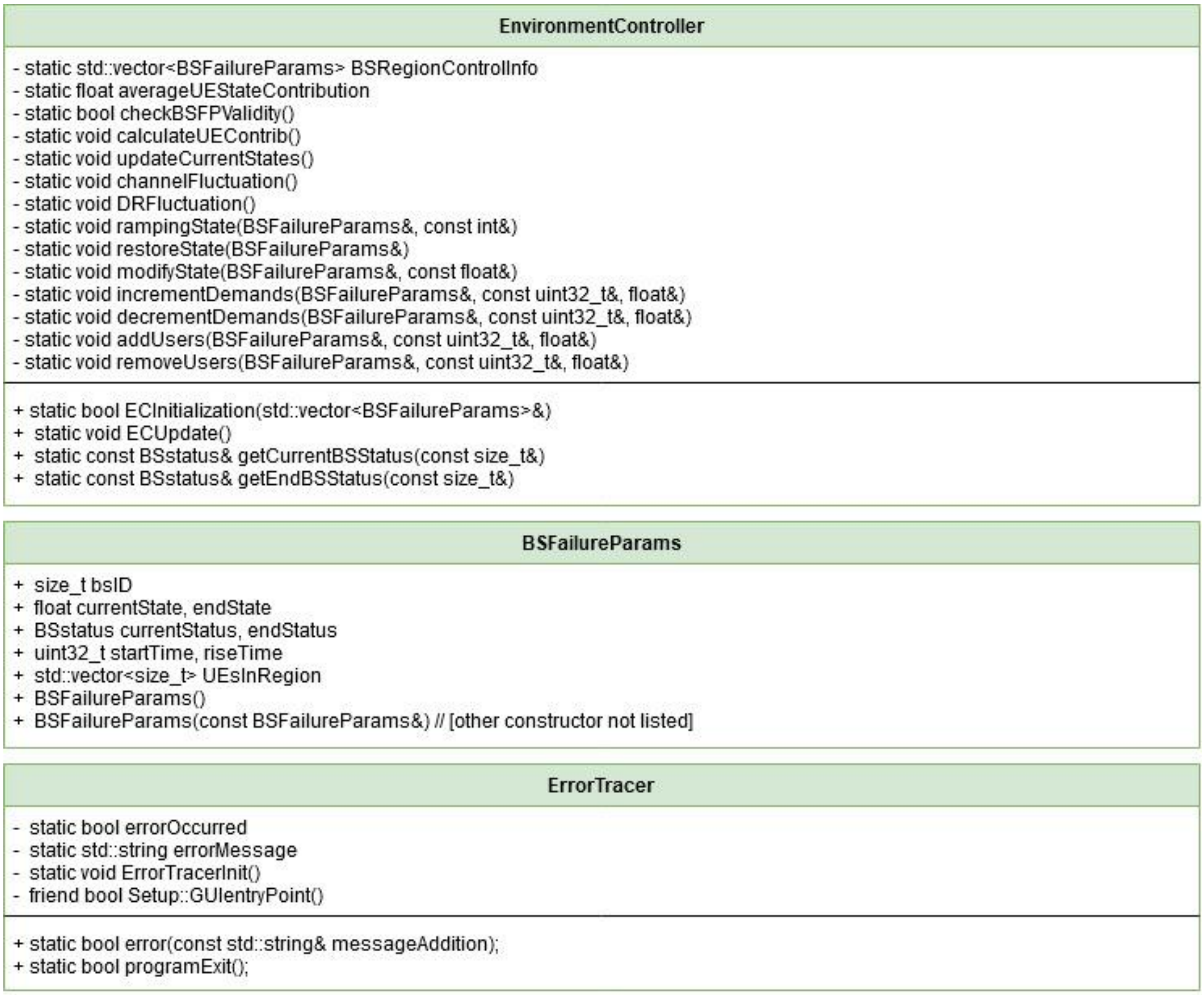
   a. Note: The class TransceiverList is appended to the Transceiver class with a grey box because it is an additional class written within the Transceiver header file.

   b. The transceiver class is used to create the transceiver objects within the network, and it allows the user to specify a transceiver / antenna angle as well as a location. The transceivers are, in a real-life system, attached to antennas within the network and this is implemented here. Within this file is the additional transceiver list class that is used to instantiate a list of transceivers. This is useful for grouping transceivers on a given antenna into a list for easy access in other portions of the program.

III. Transmission

   a. The transmission class controls the transfer of bits of data within the program. This is used to model the transferring of data from base station to user equipment.

EnvironmentController, BSFailureParams, and ErrorTracer:



```
EnvironmentController
- static std::vector<BSFailureParams> BSRegionControllInfo
- static float averageUEStateContribution
- static bool checkBSFPValidity()
- static void calculateUEContrib()
- static void updateCurrentStates()
- static void channelFluctuation()
- static void DRFluctuation()
- static void rampingState(BSFailureParams&, const int&)
- static void restoreState(BSFailureParams&)
- static void modifyState(BSFailureParams&, const float&)
- static void incrementDemands(BSFailureParams&, const uint32_t&, float&)
- static void decrementDemands(BSFailureParams&, const uint32_t&, float&)
- static void addUsers(BSFailureParams&, const uint32_t&, float&)
- static void removeUsers(BSFailureParams&, const uint32_t&, float&)

+ static bool ECInitialization(std::vector<BSFailureParams>&)
+  static void ECUpdate()
+  static const BSstatus& getCurrentBSStatus(const size_t&)
+  static const BSstatus& getEndBSStatus(const size_t&)
```

```
BSFailureParams
+  size_t bsID
+  float currentState, endState
+  BSstatus currentStatus, endStatus
+  uint32_t startTime, riseTime
+  std::vector<size_t> UEsInRegion
+  BSFailureParams()
+  BSFailureParams(const BSFailureParams&) // [other constructor not listed]
```

```
ErrorTracer
-  static bool errorOccurred
-  static std::string errorMessage
-  static void ErrorTracerInit()
-  friend bool Setup::GUIentryPoint()

+ static bool error(const std::string& messageAddition);
+ static bool programExit();
```

Figure 12: Class Diagrams 5

I. EnvironmentController

   a. The environment controller is used to "ramp up" a base station (eNodeB) into a given non-healthy state. Prior to this implementation an eNodeB specified to be congested would reach that state immediately at the beginning of the simulation, which is not accurate to real life. Currently, the program operator can specify when the eNodeB will reach congestion, the time it takes to "ramp up" and the final state of the eNodeB once the ramp up is complete.

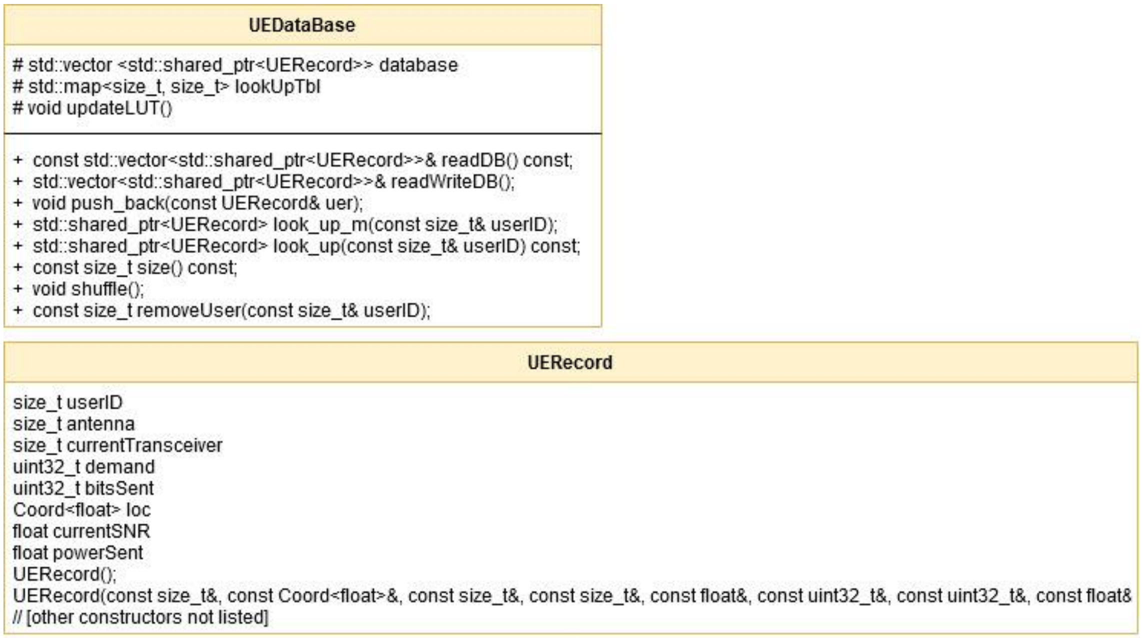
II. BSFailureParams

    a. This class is a container for the parameters specified through the GUI for the environment controller.

III. ErrorTracer

    a. This class tracks errors within the program and output an error message.

UEDataBase and UERecord:



```
UEDataBase
# std::vector <std::shared_ptr<UERecord>> database
# std::map<size_t, size_t> lookUpTbl
# void updateLUT()
+  const std::vector<std::shared_ptr<UERecord>>& readDB() const;
+  std::vector<std::shared_ptr<UERecord>>& readWriteDB();
+  void push_back(const UERecord& uer);
+  std::shared_ptr<UERecord> look_up_m(const size_t& userID);
+  std::shared_ptr<UERecord> look_up(const size_t& userID) const;
+  const size_t size() const;
+  void shuffle();
+  const size_t removeUser(const size_t& userID);
```

```
UERecord
size_t userID
size_t antenna
size_t currentTransceiver
uint32_t demand
uint32_t bitsSent
Coord<float> loc
float currentSNR
float powerSent
UERecord();
UERecord(const size_t&, const Coord<float>&, const size_t&, const size_t&, const float&, const uint32_t&, const uint32_t&, const float&
// [other constructors not listed]
```

Figure 13: Class Diagrams 6

I. UEDataBase

    a. UEDataBase is a container that holds a list of user equipment records (UERecords). This is used to access all the logs of users within a specified base station in order to receive information about a user.

II. UERecord

    a. The user equipment records is used to document attributes of a UE device, such as the user id, antenna it is attached to, the transceiver it is attached to, its demand, location, and its current SNR / power sent.

Comments:

I. The class “Simulator” was not documented due to the large size of the class, which makes it ineffective as a UML. The key idea is the simulator class is a static class that “houses” all other classes. It can be thought of as the container for the entire simulation, housing all the eNodeBs, antennas, and users.

II. UELogData was also not documented since it is simply a structure to hold user information.

III. GUIMain and GUIDataContainer were not documented since they are used to create the frontend GUI, and would therefore not fit within the backend section.

Data Rate and Power Calculations:

Maximum Data rate for a UE:

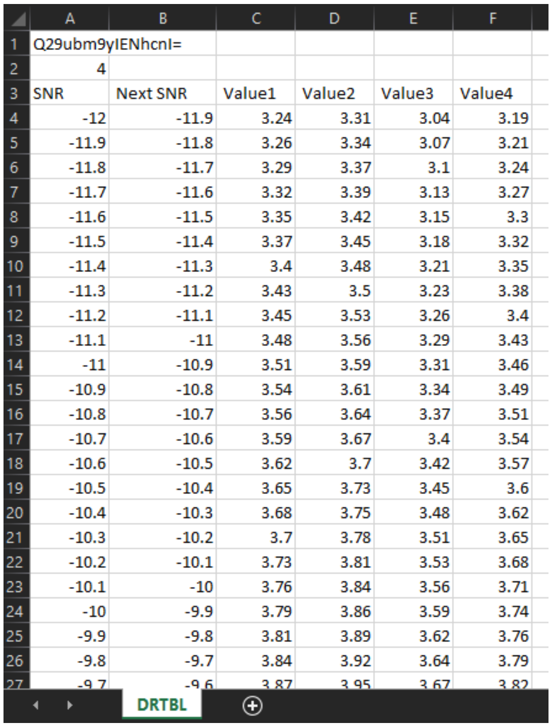
To calculate the maximum data rate a given user can receive, we first calculate the signal-to-noise-ratio (SNR) of the transmitted signal. The simulator models the SNR such that it decreases in an inverse square manner, and such that it smoothly decreases from 12db to –12db. This is calculated using the following equations:

$$1.\quad SNR_{Range} = SNR_{UpperBound} - SNR_{LowerBound}$$

$$2.\quad SNR(R^2) = \left[\frac{SNR_{Range}}{\frac{SNR_{Range} - 1}{maxDist^2} * R^2 + 1}\right] - SNR_{LowerBound}$$

Where R and maxDist are respectively the radial distance between the eNodeB and the user, and an arbitrary distance beyond which the SNR is its lower bound. The output SNR is in decibels and ranges between the upper and lower bound which are respectively 12db and –12db.

After the transmitted SNR is found, the data rate is calculated from information taken from a pre-generated table relating SNR ranges to their corresponding spectral efficiencies.



	A	B	C	D	E	F
1	Q29ubm9yIENhcnI=					
2	4					
3	SNR	Next SNR	Value1	Value2	Value3	Value4
4	-12	-11.9	3.24	3.31	3.04	3.19
5	-11.9	-11.8	3.26	3.34	3.07	3.21
6	-11.8	-11.7	3.29	3.37	3.1	3.24
7	-11.7	-11.6	3.32	3.39	3.13	3.27
8	-11.6	-11.5	3.35	3.42	3.15	3.3
9	-11.5	-11.4	3.37	3.45	3.18	3.32
10	-11.4	-11.3	3.4	3.48	3.21	3.35
11	-11.3	-11.2	3.43	3.5	3.23	3.38
12	-11.2	-11.1	3.45	3.53	3.26	3.4
13	-11.1	-11	3.48	3.56	3.29	3.43
14	-11	-10.9	3.51	3.59	3.31	3.46
15	-10.9	-10.8	3.54	3.61	3.34	3.49
16	-10.8	-10.7	3.56	3.64	3.37	3.51
17	-10.7	-10.6	3.59	3.67	3.4	3.54
18	-10.6	-10.5	3.62	3.7	3.42	3.57
19	-10.5	-10.4	3.65	3.73	3.45	3.6
20	-10.4	-10.3	3.68	3.75	3.48	3.62
21	-10.3	-10.2	3.7	3.78	3.51	3.65
22	-10.2	-10.1	3.73	3.81	3.53	3.68
23	-10.1	-10	3.76	3.84	3.56	3.71
24	-10	-9.9	3.79	3.86	3.59	3.74
25	-9.9	-9.8	3.81	3.89	3.62	3.76
26	-9.8	-9.7	3.84	3.92	3.64	3.79
27	-9.7	-9.6	3.87	3.95	3.67	3.82

Figure 14: MATLAB Generated SNR values (-12 dB to 12 dB)

The values for spectral efficiency (SE), found in columns labeled Value1-Value4, are generated using QAM 16 precoding and with the assumption of an inter-element (transceiver) spacing of half of a wavelength along with a frequency range between 20-40GHz. In order to simulate some of the real-world randomness present in any real channel, there are four different possible values for the SE that the simulator switches between randomly, i.e. columns Value1-Value4 are all valid SEs for a given SNR range and differ only due to random fluctuations in the channel as would be measured in the real world.

Knowing the SE for the channel allows us to calculate the maximum data rate for the channel which is given by:

$$DataRate = SE * Bandwidth$$

The SE has a maximum value of 8 (bits/s/Hz) and the bandwidth our simulator uses is 20MHz.

Power Transmitted and Received:

The power of the transmitted signal is proportional to the transmitted SNR as given by:

$$P_s = SNR_s * N_0$$

Here, the SNR must be converted to watts, and N_0 is a constant that represents the power of the noise present in the channel.

The power of the received signal is proportional to the power of the transmitted signal as given by:

$$P_s = \frac{P_r}{\alpha_p}$$

where α_p refers to the path-loss which in our simulator is kept at 1.2.

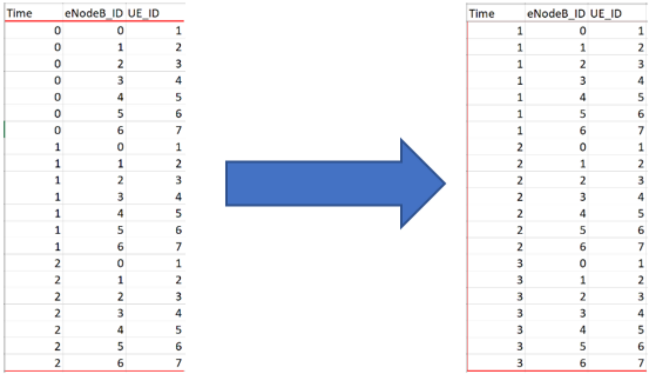
Output and Analytics (Self-Healing Process):

Storing output for analytics:

The self-healing implementation of the 5G Network Simulator first starts with detecting the condition of an eNodeB. These conditions can vary from healthy, congested from user volume, congested from user demand, and failing entirely.

In a healthy cell, the cell is operating at normal conditions, where users who are connected to the station receive the amount of data they are requesting (not experiencing lag or buffer issues). The next condition is congestion, which can be caused by a significant amount of users being connected to the base station requesting large packets (i.e.. streaming in 4k, downloading games/videos, etc.). Congestion can also be caused by a large number of users connected to a single base station, not necessarily requesting high data rates, but just having a large volume of users. In a congested state, some users will receive all of the data that they are requesting and some users won't receive any packets at all. Lastly, the final condition is "Failure" in which the eNodeB is not sending any data/packets so all users in the service area attempting to request data receive 0 packets.

Detecting these various conditions of the eNodeB is an important aspect of the self-healing process.In order to treat either a congested or failing cell, we must first be able to detect a problem. It is also important that the diagnosis of the cell is accurate as a misdiagnosis could prevent a congested or failing cell from recovering. In order to begin the process of detecting the condition of an eNodeB, we first created a storage container/buffer to store the data of all the eNodeB's for a specified amount of time. This time is specified as one of the parameters from the GUI (see section "*Setting Simulation Parameters"*). The time the user inputs is the amount of data that this buffer will store at any given time in the simulation. For a visual representation of how the buffer works, an image below is provided.



Time	eNodeB_ID	UE_ID
0	0	1
0	1	2
0	2	3
0	3	4
0	4	5
0	5	6
0	6	7
1	0	1
1	1	2
1	2	3
1	3	4
1	4	5
1	5	6
1	6	7
2	0	1
2	1	2
2	2	3
2	3	4
2	4	5
2	5	6
2	6	7

Time	eNodeB_ID	UE_ID
1	0	1
1	1	2
1	2	3
1	3	4
1	4	5
1	5	6
1	6	7
2	0	1
2	1	2
2	2	3
2	3	4
2	4	5
2	5	6
2	6	7
3	0	1
3	1	2
3	2	3
3	3	4
3	4	5
3	5	6
3	6	7

Figure 15: Buffer Visualization

For demonstration purposes, there is only one User Equipment associated with each eNodeB. In a typical simulation, there can be hundreds of User Equipment associated with each eNodeB. As seen above, the buffer/container holds the values that are bordered by the red line. The buffer initially starts with holding the data for time 0-2 (s). This indicates that our buffer will always hold 3 seconds worth of data, this buffer will then slide as time passes and hold data from 1-3 (s) as seen above. Although the buffer stores data continuously, in an effort to save time and power, the network manager will only analyze the buffer for the time specified in the GUI. For example, in the above figure although the buffer slides to hold data from 1-2 (s), the manager will only analyze the data once the 3 seconds have passed and will analyze the contents of the buffer again at 3-5 (s). In order to detect the condition of the cell, the network manager will first analyze the requested data rates with the actual data rates received by the user and determine whether or not the cell is failing. In a failing state, all of the users for a given base station in any given second will not receive any packets. In order to detect congestion, the equation we used is identified in Equation 1. The MaxData is the maximum number of packets that a single base station is able to send out to UEs in its service area in a second. ExpDR is the amount of data that an individual UE is requesting in a second, which varies randomly and depends on the congestion type. TheExpDR of all the users in a single eNodeB in a single second is then summed in order to determine the total amount of data being requested in a given second.

$$Congestion\ Level = \frac{MaxData - \sum ExpDR}{MaxDR}\ x\ 100\%$$

Equation 1: Congestion Level

Once the Congestion Level is calculated, we can determine the percentage of the MaxData of the eNodeB is still able to send out. For example, if the MaxData of a station was 1000 bits and the congestion level is 60%, then the station still has 400 bits remaining to send out. If the congestion level is at 90% or above, then the manager will determine that the cell is in congestion.

Simplified, the network manager will first look at the expected and real data rate and compare them, if these values are 0 for the entire buffer (i.e. 3 seconds) then the manager determines the cell to be failing and will initiate self-healing. Next, the network manager will look at the congestion level, if the congestion level is at the “congestion state” (as specified when setting the parameters in the GUI) or above, then the manager will determine the cell to be in “congestion” and will begin self-healing. If the congestion level is at the “alarm state” (as specified when setting the parameters in the GUI), then an “alarm” will be triggered. The alarm signifies that this particular eNodeB is approaching congestion, and although it is not in congestion, this eNodeB will not be able to assist in the self-healing process. If the network manager does not detect the failing or congested cell, then the manager determines the cell to be “healthy” and will pursue no further action.

Analysis methods:

Through the course of our project we will focus on analyzing accuracy of the simulator, and the results it produces using various self-healing techniques. It is very important that the model of the network that our simulator uses is accurate, so we will be continually examining it to see where improvements can be made to make the outputs more realistic.

Most importantly, we will be implementing self-healing techniques that allow the network to recover from failure, and so we will be looking at the data and trying to improve on the methods we implement. The key data is in the data rates; the downlink data rates in the simulator output are the biggest indicator that failure has occurred. For example, in a eNodeB that is in state of total failure, the User Equipment will not receive data, so for a given self-healing technique, the indicator of its efficacy will be how quickly and to what extent it was able to compensate for the loss of that eNodeB and restore the “Quality of Service” to the user. This will be evident in the outputs as the received data rate for a given user that was originally assigned to an eNodeB in failure will go from having zero data received to a more acceptable data rate. The same goes for an eNodeB that is merely in a state of congestion: the efficacy of the self-healing method will be evident in the overall increase in data rates amongst the users affected. While there is another group involved in this project that is using a big-data system and machine learning to analyze and improve our self-healing techniques, our analysis methods will be a bit more hands on and involve a lot of research into existing techniques and trial-and-error. It will be important for us to attempt to implement as large a variety of existing techniques into our simulator and compare them while attempting to come up with our own.

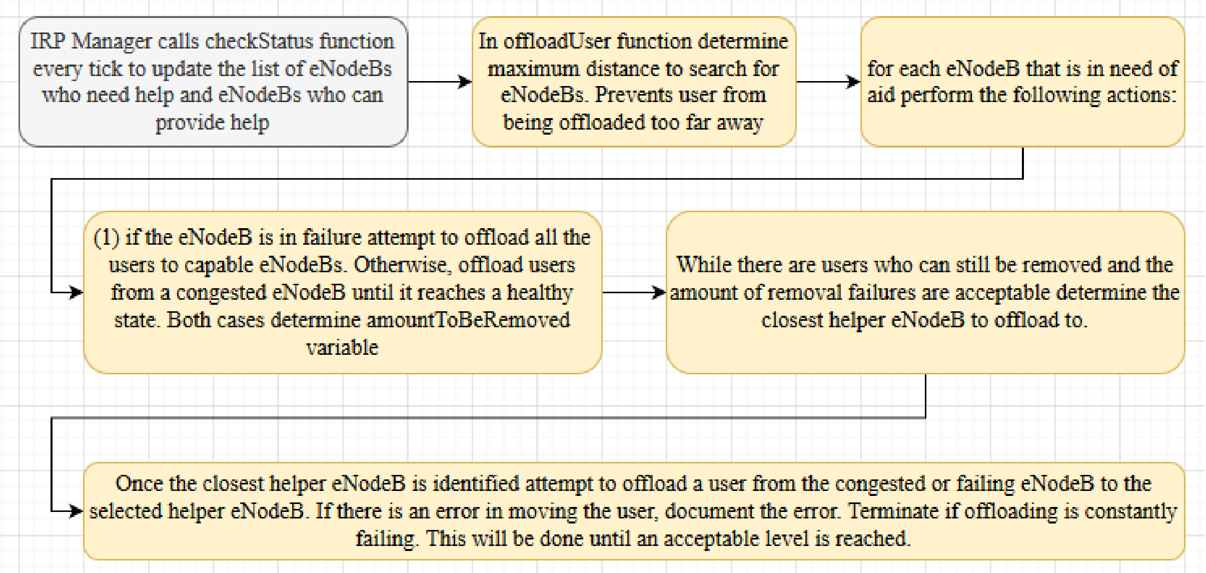
Self-Healing Algorithm

Once the condition of the cell is determined to be either failing or congested, the IRP manager will then initiate self-healing. In order to perform self-healing, we first started with a “checkStatus” function that calculates the availability of all cells in that particular system. A cell is determined to be “available” if it is not congested, failing, or in an “alarm” state. This cell will then be added to an “available list” that marks them to be used in the self-healing process to help alleviate load. If a cell is in either a congested or failing state, this function will then add them to a “disabled” list. The cells in the help list are marked so that the “offloadUser” function knows which cells to aid. If a cell is failing, in the help list they are given top priority in which the offloadUser function will prioritize serving these UEs attached to the failing eNodeB first.

The offloadUsers function takes the helplist and begins analysis on which eNodeB can be helped. This first determines the number of users needed to bring the single cell relief. It will then look at all the stations within the “available” list and calculates the distance between the coordinate where the UE is located and the eNodeB, storing the closest eNodeB as the one to offload to. Once the nearest eNodeB is identified, the system will attempt to offload the UE to the nearest antenna whose angle is facing the UE’s location in order to offload in the most effective way, one that keeps the SNR as high as possible. Once an available, nearby cell is identified this function will then assign the first UE with the highest DR requested to this cell in order to offload users from the unhealthy cell, which will reduce the load on the congested or failing eNodeB.

The users will keep being offloaded until either the unhealthy cell is brought to a healthy state or until the alarm is triggered on the aiding cell, and once the alarm is triggered (when the congestion level reaches the “alarm state” as specified in the GUI) then the aiding cell is taken off the “available” list and will provide no further relief beyond what it is already helping. At this point, the function will then assign users of the unhealthy cell to the next available cell based on the next closest base station (in order to increase the SNR and be more efficient). The cost of offloading users to healthy cells is both a higher power consumption and a lower SNR. The manager will continue to implement these self-healing procedures until either all of the cells in the network turn into a “healthy” state or until there are no more reserves for healthy base stations to aid. In either case, this algorithm will allow us to determine the improvement of the overall health of the network, which will be covered in the results section.

To summarize the self-healing algorithm, a pseudocode flow diagram is included below:



IRP Manager calls checkStatus function every tick to update the list of eNodeBs who need help and eNodeBs who can provide help

In offloadUser function determine maximum distance to search for eNodeBs. Prevents user from being offloaded too far away

for each eNodeB that is in need of aid perform the following actions:

(1) if the eNodeB is in failure attempt to offload all the users to capable eNodeBs. Otherwise, offload users from a congested eNodeB until it reaches a healthy state. Both cases determine amountToBeRemoved variable

While there are users who can still be removed and the amount of removal failures are acceptable determine the closest helper eNodeB to offload to.

Once the closest helper eNodeB is identified attempt to offload a user from the congested or failing eNodeB to the selected helper eNodeB. If there is an error in moving the user, document the error. Terminate if offloading is constantly failing. This will be done until an acceptable level is reached.

The grey box describes the purpose of the checkStatus function within the IRPManager class while the yellow box describes the offloadUser function that is used within the IRPManager to appropriately offload users.

Results / Conclusion

Using the 5G Network Simulator as our ecosystem, we were able to successfully verify that the IRP manager is performing self-healing as intended. Shown in figure 17 below are graphs generated using output from the CSV files, which aim to visualize self-healing within the network. Note that in the scenario below 1 eNodeB (marked in blue) is congested while the others remain healthy.

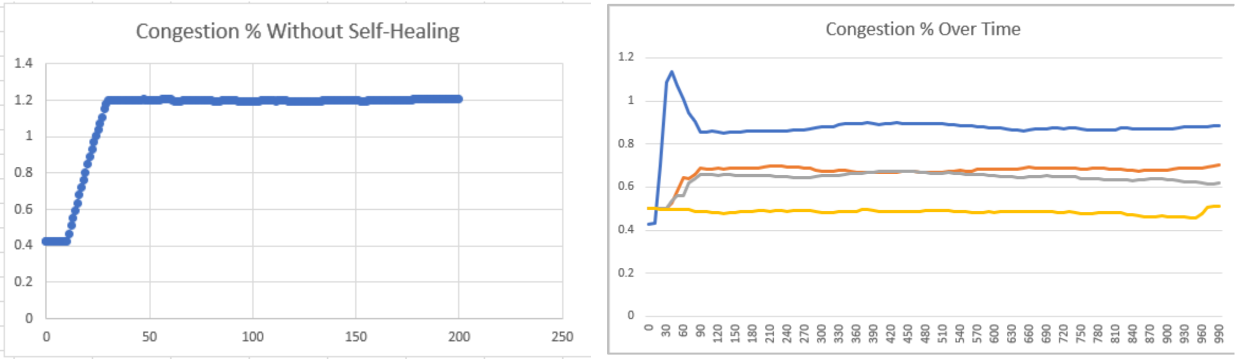


Figure 17: Congestion levels before and after Self-Healing

The graph on the left shows how the network congestion over time would stabilize at 1.2 (120%) for the congestion without the intervention of self-healing. On the right, the IRP manager is called to perform self-healing on the eNodeB that needs aid. We can see that the eNodeB in need, denoted by the blue line, almost reaches 120% congestion, but is reduced to a healthy level using self-healing. We can also see that the two neighboring eNodeBs (the grey and orange lines) have increased since they “claimed” the users that needed to be offloaded. Nonetheless, the overall health of the network is now healthy since none of the eNodeBs are congested or failing, which is the ideal scenario.

The scatterplot in figure 18 displays the same information as figure 17 in a more intuitive manner.

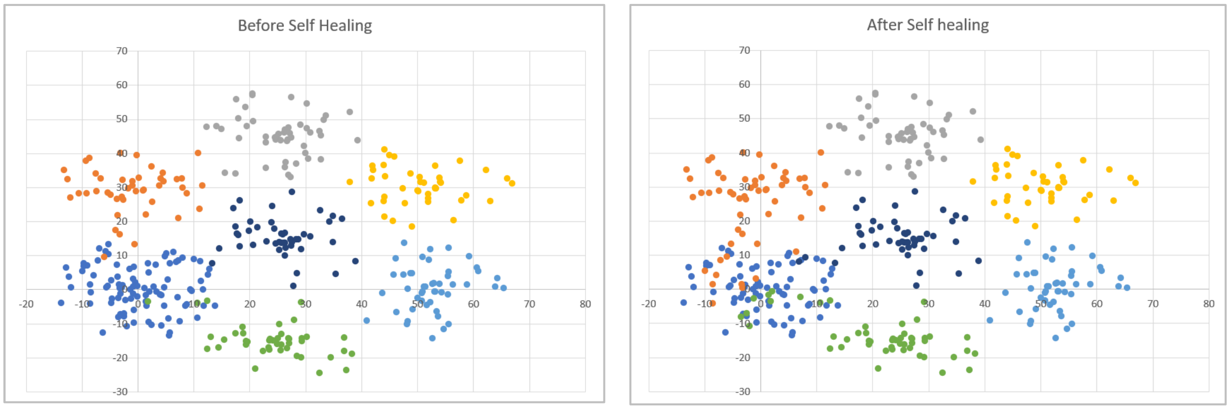


Figure 18: Scatter plot before and after self-healing

Each point on the scatter plot above indicates the location of the User Equipment and the color of the point corresponds to the eNodeB that it is associated with. The blue cluster of points in the bottom left of the graph above represents the eNodeB in need of aid, while the orange, navy blue, and green clusters represent the neighboring cells for our configuration. It is apparent that after self-healing the eNodeB in need of aid had some of its users offloaded to the orange and green eNodeBs around it. This is especially visible when comparing the two images side-by-side since there are fewer blue points and orange and green points overlap in that particular cluster.

Appendix A: Standards, Constraints, Project Planning, Task Definition

Constraints

Our project of 5G Self-Healing Networks fortunately does not entail very many constraints since the majority of our project is software based. The monetary constraints that are present are just the licensing of MATLAB, however our project consists primarily of programming in C and C++ which we have licenses for on Visual Studio as well as a Linux Virtual Machine. There is also an accessibility constraint since this is a group project including six students all working off the same code base, which unfortunately there isn't a free software working similar to google drive where we can all make live time edits to the code.

Engineering Standards

Since our project is only a code-based simulation, there aren't any standards to be adhered to. The only standards we must adhere to is making sure that our simulation meets the 5G standards of having 1ms latency, upwards of 1Gb/s download speed, and that our network is able to successfully self-heal.

Societal Impacts

Global Impact:

The global impact of our project is largely positive. The implementation of self-healing networks will decrease the amount of time that cell-service is out, and will help when certain stations/areas are flooded with users. Rather than having a person go out to inspect a tower for minor fixes, the network will be able to heal itself restoring it to full functionality. Also, in times of emergency such as a natural disaster or terrorist attack everyone in the affected area contacts emergency services. In situations such as these, cell towers can often become flooded with users and render the station inoperable. However, the implementation of self-healing networks will allow for nearby stations to alleviate some of the load so that people in the affected area will have service and still be able to contact emergency services.

Economic Impact:

The potential economic impact of future commercial use is primarily positive. The development of self-healing networks reduces the cost to operate base stations as now operators will have less of a need to troubleshoot problems in base stations. Companies utilizing these algorithms save money which allows for reduced service costs for customers. Since 5G creates the need for significantly more infrastructure, if operators needed to fix these massive amounts of stations for any small problem, the cost to use 5G would increase dramatically.

Ethical and Professional Responsibility Judgement

Two issues that pertain to our project is that with the advent of 5G technology, the vision of fully autonomous vehicles become much realistic and also the infrastructure needed creates environmental issues. Since 5G allows for near instant transmission speeds, fully autonomous vehicles will now be more realistic as cars can communicate with each other. Many people are against the creation of fully autonomous vehicles since this takes the control out of the people. In accidents involving autonomous vehicles, no one knows who to blame since it can be the driver, engineer designing the software of the vehicles, or the car company that sold the vehicle. However, the benefits of 5G greatly outweighs this issue as now people who are unable to drive on their own are now able to travel by having the car drive for them. For the second issue of the increased infrastructure, people have issues with the infrastructure since trees can be removed in order to build base stations, and also cell towers aren't as "aesthetically pleasing" decreasing property value in urban areas. However, these base stations are very important especially in urban areas since these are densely populated areas, the likelihood of base stations being overloaded is much more likely than that of a rural area. So, with the self-healing network, the reliability of self-service is significantly improved allowing for people in emergency situations to have constant service to contact emergency services.

Project Planning

The end goal of this project is to have done all the necessary research, written the paper, created the poster, and added a self-healing portion of the code that parses the data, interprets it, and generates a solution. Every member also has access to a VM that we use to simulate on a standardized platform. The current simulator has the ability to simulate multiple hexagonal cells, with multiple users on each cell, and output a CSV file with data describing the interaction between each node and its users. It also has multiple possible states for cells, such as when they are overburdened, or if a node goes down. The self-healing of the 5G simulator would be able to find the optimal solution of redirecting traffic if a node went down. The idea is to have the process be completely automated.

The simulator models a real network by calculating a number of parameters that characterize the channel and real-world constraints. It takes an arbitrarily formulated SNR value and then accesses a pre-generated lookup table to come up with the spectral efficiency of the channel. From this the maximum possible data rate and transmission power can be calculated based off of the bandwidth and desired received power. The pre-generated look up table was created using a MATLAB script designed to produce values for a network utilizing beamforming MIMO antennae.

The current simulator simply takes input parameters from the user, and using the MATLAB portion of the code it generates a virtual environment and executes the simulation. However, it isn't capable of self-healing at the moment, the channel parameter calculations need to be finalized, and the simulator as a whole needs to be made more efficient. We also need to explore other methods of performing the channel calculations and since this portion of the simulator largely characterizes the technologies being simulated, it will be important that this process remain modular so that different technologies can be simulated easily.

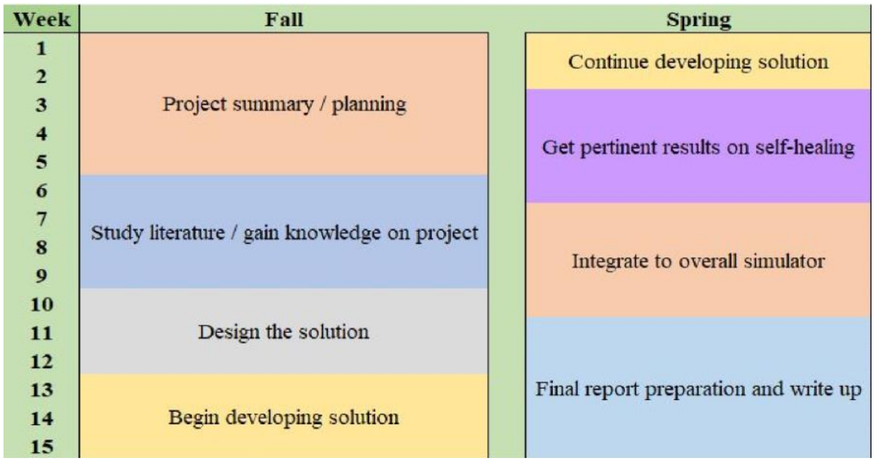
To start the self-healing portion of the project, we most likely will find an existing working algorithm that works for our code. Another idealization is that we get the simulator working more efficiently, or finding some way of managing the data that runs faster. The more efficient we can make the simulator, the more tests we can run to measure the efficacy of the self-healing. It's possible that we'll find multiple algorithms online, and we will compare them and see which produces the best results.

Task Definition:

Our project entails the development of a 5G simulator with a main focus on developing optimal self-healing strategies for autonomous network management through the use of beamforming. For software requirements, the 5G network simulator is written in C++. In addition, MATLAB is used to describe various mathematical calculations such as channel conditions and modeling transceiver power usage, which are incorporated into the C++ program for real-world simulation data.

There are two main study variables that this project is observing: self-healing using beamforming along with the idea of a system recovering an eNodeB when it is malfunctioning. Beamforming is a technique that focuses a signal towards a specific receiving device, rather than having the signal spread out in all directions. This results in a more direct connection, which is faster and more reliable than the standard way of receiving and transferring signals. For this project, beamforming will be the main point of observation to see if beamforming can help assist in self-healing. Using beamforming, this project hopes to use certain algorithms to employ self-healing techniques on eNodeBs to help other eNodeBs assist a nearby eNodeB that is failing or congested. Furthermore, self-healing will be observed to see if the method can be applied autonomously for the eNodeBs to use and communicate with one another to help assist each other.

Our Gantt chart for the project serves as an outline of our project during the 2019 – 2020 school year:



Week	Fall	Spring
1	Project summary / planning	Continue developing solution
2		
3		Get pertinent results on self-healing
4		
5		
6	Study literature / gain knowledge on project	
7		Integrate to overall simulator
8		
9		
10	Design the solution	
11		Final report preparation and write up
12		
13	Begin developing solution	
14		
15		

Appendix B: Running the GUI on Linux (Ubuntu distribution)

Prerequisites:

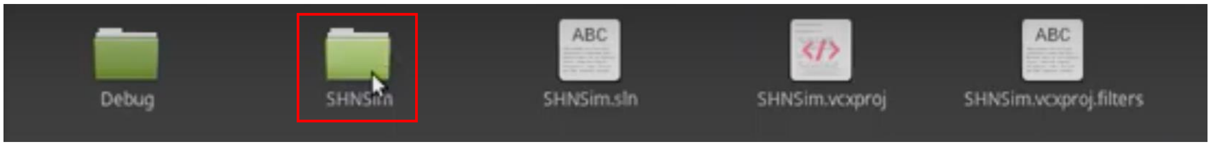
In order to run the Graphical User Interface several libraries are needed. The commands to install them on an Ubuntu operating system are as follows:

- `sudo apt-get update`
- `sudo apt-get upgrade`
  - Should be done during a fresh install of an OS prior to installing the other libraries.
- `sudo apt install make`
  - The "make" library is needed to run the commands to easily compile multiple class files into one executable. Without this, the files would need to be manually compiled and linked.
- `sudo apt install build essential`
  - This installs the necessary C/C++ compilers that are needed to compile the codebase.
- `sudo apt-get install gtk+-3.0`
  - The GUI was implemented with this library, which makes this needed to both program and view the interface

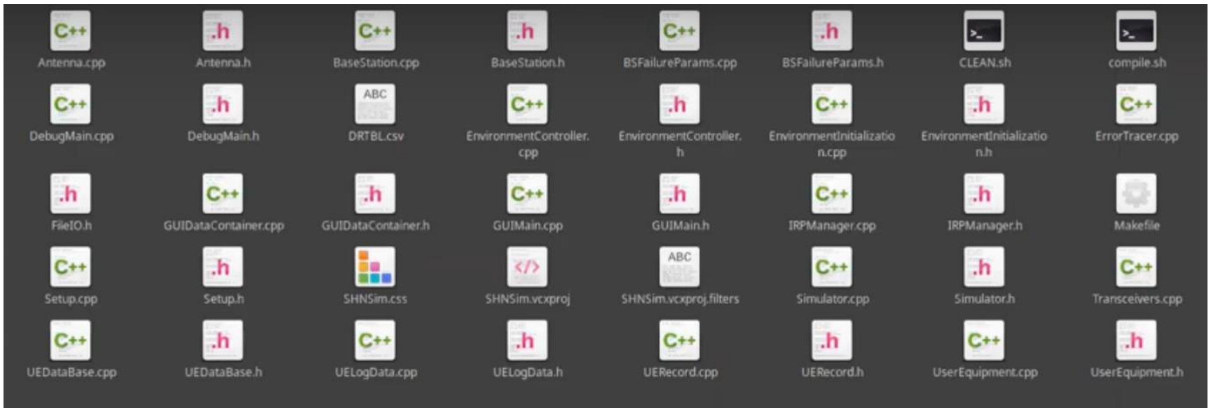
Guide:

Please note, there is a video on Youtube from Omar Naffaa who was a member working on the 5G Self-Healing Network Simulator in the 2019-2020 school year that is a tutorial on how to run the GUI on a Linux OS. The link to that video is Navigating the 5G SHN Simulator in Linux. This provides a clear explanation of how to run the simulation. This document is a quick-start guide on how to run the simulation using the GUI in order to create the .csv file containing the results of the simulation.

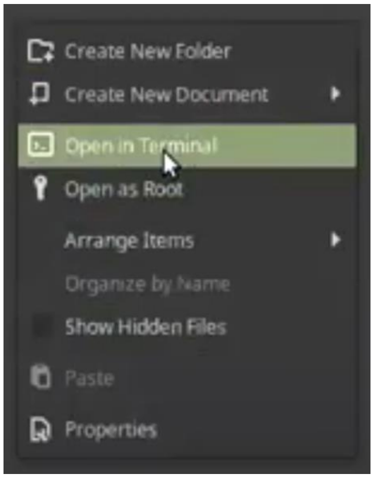
1. Save the Self-Healing Network Simulation folder onto a Linux Desktop. The simulation folder containing the code can be either retrieved from the Github repository or from the CPP Virtual Machine.
2. Once you have opened the folder, click on "SHNSim" to open the folder containing the code



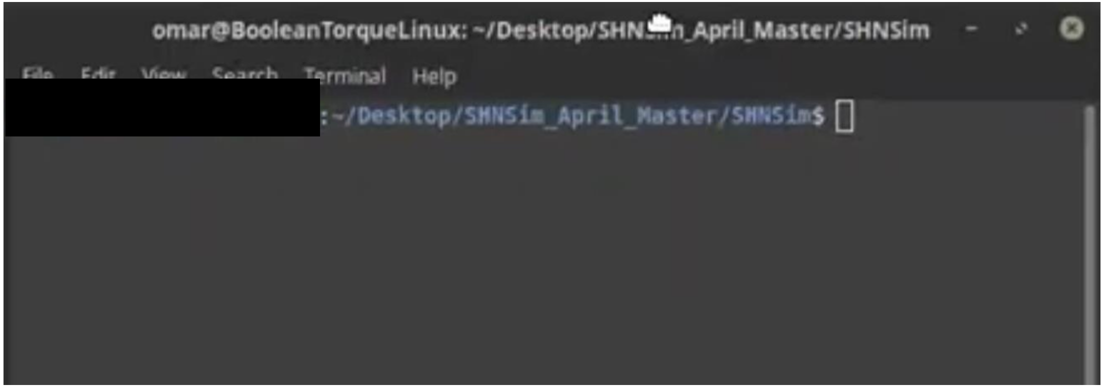
3. The folder should open to show all the code that will run the simulation as shown below. Please note this does not include all files, but rather some of the files used to run the simulation.



4. Right click a blank space inside of the folder (Don't right click a file otherwise you will just get options to do with that file) and click on "Open in Terminal" as highlighted below



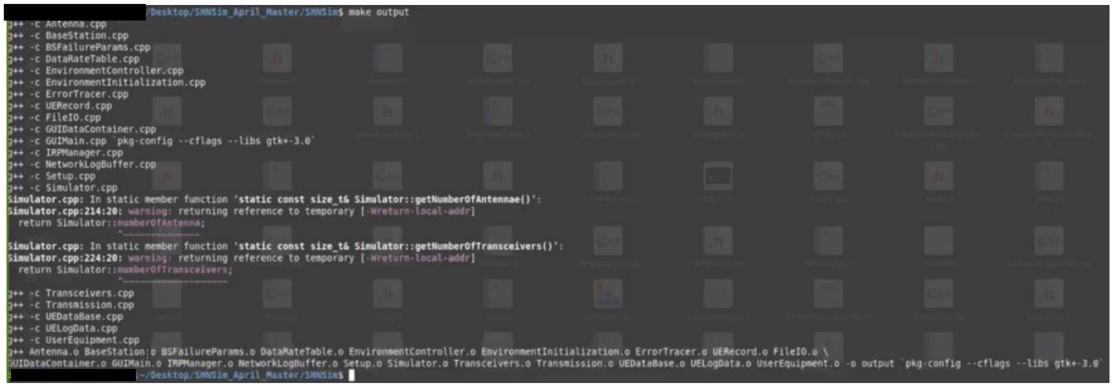
5. The Linux Terminal will open as shown below. (The name of the owner has been redacted for privacy reasons)



```
omar@BooleanTorqueLinux: ~/Desktop/SHNSim_April_Master/SHNSim
```

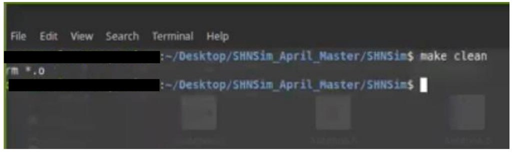
```
[redacted]:~/Desktop/SHNSim_April_Master/SHNSim$ 
```

6. The first command should be "make output" and then click enter. The image below demonstrates this and then the corresponding about on the terminal window. This command will create objects for the simulation and link it to an executable entitled "output".



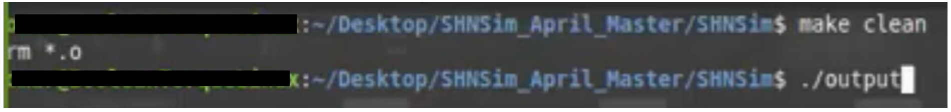
```
[redacted]/Desktop/SHNSim_April_Master/SHNSim$ make output
g++ -c Antenna.cpp
g++ -c BaseStation.cpp
g++ -c BSFailureParams.cpp
g++ -c DataRateTable.cpp
g++ -c EnvironmentController.cpp
g++ -c EnvironmentInitialization.cpp
g++ -c ErrorTracer.cpp
g++ -c UERecord.cpp
g++ -c FileIO.cpp
g++ -c GUIDataContainer.cpp
g++ -c GUIMain.cpp `pkg-config --cflags --libs gtk+-3.0`
g++ -c IRPManager.cpp
g++ -c NetworkLogBuffer.cpp
g++ -c Setup.cpp
g++ -c Simulator.cpp
Simulator.cpp: In static member function 'static const size_t& Simulator::getNumberOfAntennae()':
Simulator.cpp:214:20: warning: returning reference to temporary [-Wreturn-local-addr]
  return Simulator::numberOfAntenna;
                    ^~~~~~~~~~~~~~~

Simulator.cpp: In static member function 'static const size_t& Simulator::getNumberOfTransceivers()':
Simulator.cpp:224:20: warning: returning reference to temporary [-Wreturn-local-addr]
  return Simulator::numberOfTransceivers;
                    ^~~~~~~~~~~~~~~~~~~~

g++ -c Transceivers.cpp
g++ -c Transmission.cpp
g++ -c UEDataBase.cpp
g++ -c UELogData.cpp
g++ -c UserEquipment.cpp
g++ Antenna.o BaseStation.o BSFailureParams.o DataRateTable.o EnvironmentController.o EnvironmentInitialization.o ErrorTracer.o UERecord.o FileIO.o \
GUIDataContainer.o GUIMain.o IRPManager.o NetworkLogBuffer.o Setup.o Simulator.o Transceivers.o Transmission.o UEDataBase.o UELogData.o UserEquipment.o -o output `pkg-config --cflags --libs gtk+-3.0`
[redacted]:~/Desktop/SHNSim_April_Master/SHNSim$
```

7. The next command is “make clean”. This command will remove the object files in order to save space. These object files are no longer needed since the “output” file/executable has already been created.



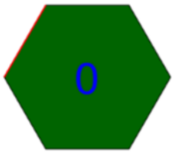
```
[redacted]:~/Desktop/SHNSim_April_Master/SHNSim$ make clean
rm *.o
[redacted]:~/Desktop/SHNSim_April_Master/SHNSim$
```

8. The final command to compile the code is “./ output”. This command will compile the output file and begin the GUI

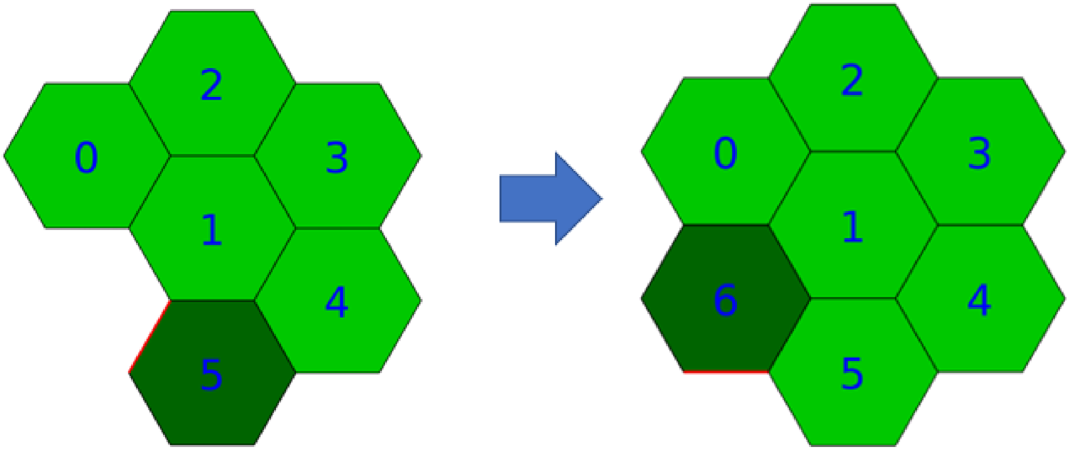


```
[redacted]:~/Desktop/SHNSim_April_Master/SHNSim$ make clean
rm *.o
[redacted]:~/Desktop/SHNSim_April_Master/SHNSim$ ./output
```

9. Assuming the code compiled from the above step, the next window that pops up will be the GUI shown below. The number indicates the base station ID. The base station ID can’t be changed, however the status of the base station can be changed by left-clicking the base station ID. A green cell indicates a normal/healthy cell, a yellow cell indicates an “unhealthy/congested” cell due to a high volume of users, an orange cell indicates an “unhealthy/congested” cell due to high data demands, and a red cell indicates a cell in “failure”.



10. Shown in the step prior is a hexagon that represents a single base station. However, we typically wish to simulate a system of base stations (since a system is needed in order to perform self healing). In order to create more base stations, hover the mouse in the side that you wish to create an additional base station and left-click on the mouse. A red line will appear on the side of the hexagon that shows where the base station will be placed as shown in the first image below. As a result of left-clicking where the red line is indicated, the next image shows the base station that is built. The cells that are dark green is the base station that is selected. In order to change the selected base station, left-click anywhere on a base station (except for the number since clicking on a number will change the status of the base station).



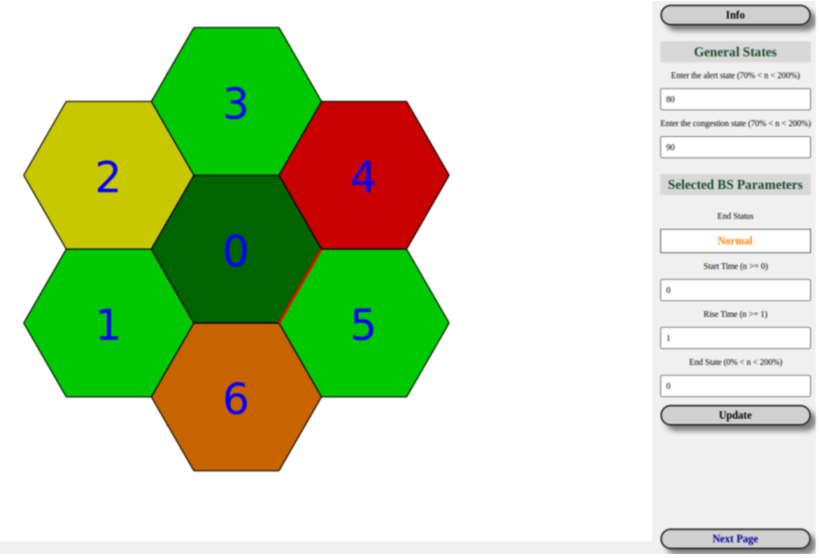
11. In case a base station was created by accident, they can be deleted simply by right-clicking anywhere on the base station that will be deleted.
12. After the base stations have been placed in the position/setup desired, the next step is to fill in the various simulation parameters. The image below is an example of a simulation with all 4 network conditions represented. The boxes on the right are parameters for the simulation and will be explained below.
    a. **Enter the alert state (70%<n<200%):** This box allows for the user to define when the base station will trigger the "alarm". In our simulator, the metric we use to differentiate between a "healthy" and a "congested" cell is represented by the equation: % healthiness = $\frac{BS\ Max\ Data\ Rate}{\sum UE\ data\ rates}$. This equation essentially takes a Base Stations Maximum data that it is able to send out per second, and divides it by the sum of all the data rates from the UEs connected to that particular base station in that same second. By selecting "n" in this step to be a particular value between 70-200%, this allows the user to select at which percentage the base station will trigger the "alarm". The alarm represents when a base station is approaching a "congestion" state and is significant because once a base station is in an "alarm" state, it no longer has the capacity to aid neighboring cells in the self-healing process. It essentially tells the network manager that although it is not in congestion, it is approaching congestion and will not be able to help since helping neighboring cells could possibly put it into a congestion state. Please note that the value input for the alert state must be lower than that selected in the following box, "congestion state".
    b. **Enter the congestion state (70%<n<200%):** This box allows for the user to define when a base station has entered a "congestion" state. This is important since once "% healthiness" (as described in 12.a) reaches this value set by the user, the IRPManager will begin to implement self-healing. The self-healing algorithm will continue until either the congestion level of the cell is below this parameter, or is no available help from nearby cells.

c. **End Status:** This is the status of the cell. This will be either “Healthy”, “User Congested”, “Demand Failure”, “Failure” depending on what the user selected when creating the hexagons/base stations.

d. **Start Time (n>=0):** Start time is the time (in seconds) at which the selected base station will begin ramping up to the “End Status” as defined above. The time it takes to reach the “End Status” depends on the next parameter, “Rise Time”. Please note that all cells will start off as healthy until the end of the Rise Time.

e. **Rise Time (n>=1):** Rise time is the time (in seconds) that it will take to reach the “End Status”. For example, if the start time is 50 and the rise time is 10, then the selected cell will be healthy until t=50 seconds, at which point the base station will start to ramp-up/ramp-down to enter the “End Status”. At t=60 seconds, the base station will be in the “End Status” state.

f. **End State (70%<n<200%):** This parameter pertains to a base station that is selected to be “congested”. This once again pertains to % healthiness described in 12.a, except now this is the value that the base station will reach. For example, if alarm state is 70% and congestion state is 80% and the end state is 75%, then the base station will trigger the alarm, but not enter congestion so self-healing will not be performed. However, if the end state was 100%, then the self-healing algorithm will be performed since the base station will ramp up to 100% and the self-healing algorithm will continue to be performed until the base station healthiness is either below the congestion state value or until nearby base stations are no longer able to help.



General States

Enter the alert state (70% < n < 200%)

80

Enter the congestion state (70% < n < 200%)

90

Selected BS Parameters

End Status

Normal

Start Time (n >= 0)

0

Rise Time (n >= 1)

1

End State (0% < n < 200%)

0

13. Next are the Network Parameters, Simulation Parameters, and Self-Healing Parameters as shown in the window below. Each of the parameters are explained below

a. **BS Side Length (5<n<20):** This is the base station side length, increasing the length of the base station corresponds with increasing the distance between the base stations and the user equipments. The higher the value of the side length, the lower the SNR ratio is and will correspond with an increase in power needed in order to be able to send the beams further distances.

b. **Number of Transceivers (80<n<200):** This is the number of transceivers per antenna that will connect to the User Equipments. Since 5G utilizes Massive MIMO (Multiple Input, Multiple Output) technology, there are lots of transceivers per antenna that will then connect to each individual UE.

c. **Number of Antennas (1<n<6):** This is the number of antennas per base station. Since each base station provides coverage for 360° around that specific cell. The field that each antenna provides coverage for depends on the number of antennas. For example, if there are 3 antennas, then each antenna will provide coverage/cell data to 120°, and if there are 4 antennas then each cell will cover 90°.

d. **Enter UEs per Antenna [normal BS] (1<n<40):** This parameter allows the user to select the number of UEs per antenna for a healthy base station. This number is increased/decreased depending on how much data you want to show in the resulting .csv file once the simulation is ran. Please note, this is only for healthy cells since once a cell is set to congestion, depending on the type of congestion the simulator may add UEs in order to enter the cell into a state of congestion.

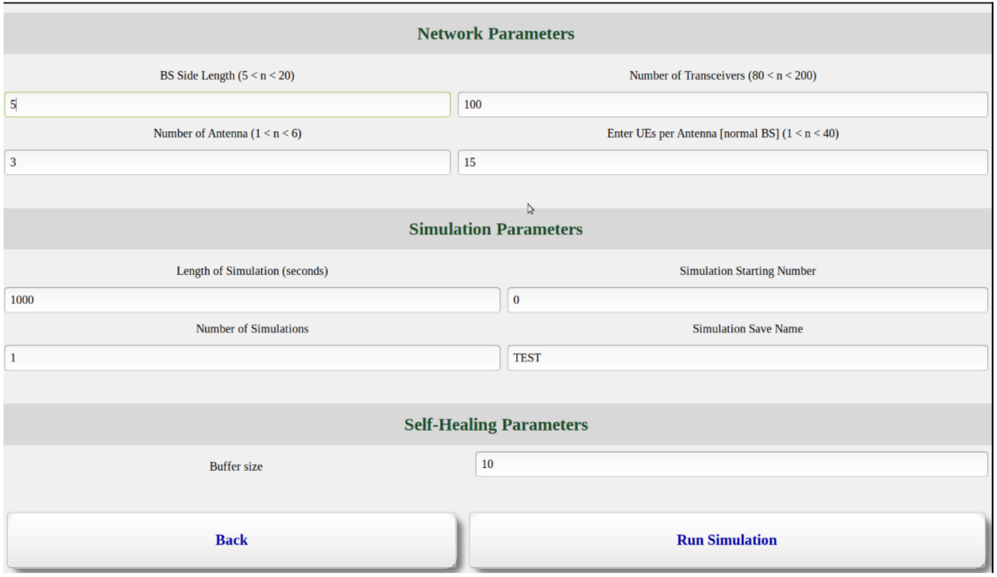
e. **Length of Simulation (seconds):** This parameter allows for the user to indicate how long they would like the simulation to run for. This parameter is entered in seconds and the longer the simulation is, the longer the simulation will take to create the results and the more space the results file will occupy.

f. **Simulation Starting Number:** This number should be entered as "0" unless you are continuing a simulation that was otherwise corrupted/stopped. For example, if you ran 4 simulations and for some reason you had to stop at the 2nd simulation because your computer shut off, you can enter "3" to show that this is the starting simulation number

g. **Number of Simulations:** This parameter allows you to enter how many times you'd like to run the simulation with the same network conditions. Since there plenty of elements of randomness throughout the code, even if two simulations were set up the exact same way, different results would be output. So this allows for you to run multiple simulations with the same parameters in case you wanted to compare results.

h. **Simulation Save Name:** This is the name of the results file and can be named whatever you'd like and will be in the form of a .csv file. So if "TEST" is entered, then the output file found in the simulation folder will be "TEST.csv".

i. **Buffer Size:** This parameter is specifically related to the self-healing algorithm. The buffer is a storage container that holds all information for each UE in each base station for the specified amount of time in seconds. The buffer will initially start with holding the data for the time specified in the parameter, for example let's say we entered 3 so the buffer will hold date for time 0-2 (s). This indicates that our buffer will always hold 3 seconds worth of data, this buffer will then slide as time passes and hold data from 1-3 (s). Although the buffer stores data continuously in an effort to save time and power, the network manager will only analyze the buffer for the time specified. For example, in the above figure although the buffer slides to hold data from 1-2 (s), the manager will only analyze the data once the 3 seconds have passed and will begin analysis again at 3-5 (s).



Network Parameters

BS Side Length (5 < n < 20): 5

Number of Transceivers (80 < n < 200): 100

Number of Antenna (1 < n < 6): 3

Enter UEs per Antenna [normal BS] (1 < n < 40): 15

Simulation Parameters

Length of Simulation (seconds): 1000

Simulation Starting Number: 0

Number of Simulations: 1

Simulation Save Name: TEST

Self-Healing Parameters

Buffer size: 10

14. The simulation will take some time to complete depending on the length of the simulation, number of base stations, and number of UEs. Upon successful completion the results of the simulation should be in a .csv file with the title that was put in the “Simulation Save Name” parameter. This .csv file should be located in the same folder where all of the code is stored. Depending on the length/size of the simulation, there may be multiple .csv files but they will all correspond to the same simulation file.

References

[1] J. J. e. al, "Smart Small Cell with Hybrid Beamforming for 5G: Theoretical Feasibility and Prototype Results," *IEEE Wireless Communications,* vol. 23, no. 6, pp. 124-131, December 2016.

[2] N. S. a. A. R. M. Alias, "Efficient Cell Outage Detection in 5G HetNets Using Hidden Markov Model," *IEEE Communications Letters,* vol. 20, no. 3, pp. 562-565, March 2016.

[3] H. F. a. A. I. A. Asghar, "Self-Healing in Emerging Cellular Networks: Review, Challenges, and Research Directions," *IEEE Communications Surveys & Tutorials,* vol. 20, no. 3, pp. 1682-1709, 2018.

[4] O. Scheit, "Self-Healing in Self-Organizing Networks," *Seminar Innovative Internettechnologien und Mobilkommunikation,* pp. 175-181, 2014.

[5] S. K. B. S. L. B. a. M. K. J. Ali-Tolppa, "SELF-HEALING AND RESILIENCE IN FUTURE 5G COGNITIVE AUTONOMOUS NETWORKS," *2018 ITU Kaleidoscope: Machine Learning for a 5G Future,* pp. 1-8, 2018.

[6] F. N. a. S. Haryadi, "Simulation of the fault management with Self Healing mechanism (case study: LTE Network in Banda Aceh Area)," *2016 10th International Conference on Telecommunication Systems Services and Applications (TSSA),* pp. 1-6, 2016.

[7] W. R. e. al, "Millimeter-wave beamforming as an enabling technology for 5G cellular communications: theoretical feasibility and prototype results," *IEEE Communications Magazine,* vol. 52, no. 2, pp. 106-113, February 2014.

[8] M. Passoja, "RCRWirelessNews," 12 September 2018. [Online]. Available: https://www.rcrwireless.com/20180912/5g/5g-nr-massive-mimo-and-beamforming-what-does-it-mean-and-how-can-i-measure-it-in-the-field. [Accessed 2 October 2019].

[9] D. W. a. C. Dewar, *Understanding 5G: Perspectives on future technological advancements in mobile,* GSMA Intelligence, Walbrook, 2014.

[10] P. B. N. G.-P. Y. W. a. R. W. H. A. Klautau, "5G MIMO Data for Machine Learning: Application to Beam-Selection Using Deep Learning," *2018 Information Theory and Applications Workshop (ITA),* pp. 1-9, 2018.

[11] K. D. a. R. D. Gitlin, "5G green networking: Enabling technologies, potentials, and challenges," *2016 IEEE 17th Annual Wireless and Microwave Technology Conference (WAMICON),* pp. 1-6, 2016.

[12] A. Trehan, *Algorithms for Self-Healing Networks,* 2018.

[13] B. L. X. Z. Z. Y. a. M. Y. Y. Xie, "SpringerLink," 11 October 2018. [Online]. Available: https://link-springer-com.proxy.library.cpp.edu/article/10.1007/s11276-018-1846-5. [Accessed 25 September 2019].