5G Network Simulator Report

Version 1.0 - December 2025
San Diego State University - CS576

Introduction

This project presents an interactive 5G Network Simulator designed to help users visualize and experiment with the fundamental behaviors of modern cellular systems. Key features include full IPv4 support and a simple CLI backend for non GUI experiments. The simulator combines a real-time graphical interface with an underlying event-driven model that captures key aspects of 5G operation, including tower coverage, mobility, handovers, outages, and data transmission. By allowing users to place towers and UEs, adjust network conditions, and observe system responses, the tool provides an accessible way to study how network topology and radio characteristics affect performance. Although the simulator abstracts many complexities of the full 5G standard, it preserves the core mechanisms necessary for educational exploration. This report describes the simulator's goals, features, modeling assumptions, user instructions, and experimental results to give a complete understanding of how the system operates and how it can be used for analysis.

Network Simulator Main Goals

In this project, we aim to simulate:

- Full IPv4 integration in a dynamic and noisy 5G scenario.
- Congestive network topologies, where traffic routing is dynamic.
- Seamless UE transitions between towers for real world connectivity scenarios.
- Noisy channel transmission and reliable IP packet delivery
- Real-world scenarios such as random dropout and cell outages.
- Scenarios such as tower outages to test capability of a network with limited cells.
- Full network routing capabilities in high throughput, multi-access topologies.
- High user customizability of devices for planning of real world 5G network systems.
- Simple GUI interface for simplistic user interaction and planning.

Features (specs + features of the GUI)

Hexagonal Canvas: The simulator uses a scalable hexagonal grid to represent the physical coverage layout typical in cellular networks.

Key characteristics-

- Automatic grid snapping ensures towers align correctly
- Hex-cells represent potential tower placement positions.

Tower Status Control: Each tower supports multiple operational states

- **ACTIVE:** Fully operational; participants in coverage and routing
- **DISABLED:** Removed from the active network; UEs detach automatically
- **OUTAGE:** Temporarily non-operational due to simulated failure

Tower Properties: GUI allows viewing and changing of information regarding the selected tower, such as…

- IP address, grid position, operating status
- Real-time throughput and internal buffer usage
- Manually activating, disabling, or triggering an outage
- Connecting a tower to another tower (backhaul links)
- Disconnecting tower-to-tower links
- Deleting towers from the canvas entirely

Backhaul Network visualization:

- All tower-to-tower connections are shown with blue dotted lines
- Links remain visible at all times and automatically update when towers move or are reconfigured.

Placement and Mobility:

- Users can place UEs anywhere on the canvas
- UEs are draggable in real time, supporting mobility testing and live handovers.
- While dragging, the UE's tower selection and connection line update dynamically

UE Property Panel: left-clicking a UE opens an interactive window showing…

- UE IP address
- Current position
- Connected tower and frequency band
- Code rate (LDPC simulation)
- Maximum achievable data rate
- Actual throughput (derived from per-timestep transmission rates)

Transmission Control: Each UE can be configured to transmit data to any other UE using...

- **Fixed Mode:** send a specific number of bytes per timestep
- **Random Mode:** send 1-65,535 bytes per timestep
- **Max Mode:** Transmit at the maximum at max_data_rate × code_rate

Real-Time Simulation Controls: The top control panel provides global controls over the simulation engine

- Start SIM begins the real-time simulation loop
- Stop SIM pauses all updates

Global Outage and Disablement:

- Simulate total outage: all towers fail for a user-specified number of steps.
- Disable All Towers: Immediately removes all towers from the network.
- Reset UEs: clears all UE transmission buffers and resets their properties.

Frequency Band Visualization: The GUI supports dynamic visualization of cellular coverage for different 5G bands.

- High Band: small radius, high throughput
- Mid Band: moderate radius
- Low Band: Large radius, lower throughput
- Switching bands updates: tower range, UE connection logic, and visual representation of band-specific coverage

Noise and interference Simulation: A toggleable noise model allows realistic testing of channel reliability

- Noise is based on a distance-dependent loss curve
- Code rate directly affects packet drop probability
- Both UE and tower layers have built-in dropout functions

Tower-UE Connection Visualization: each UE actively displays its serving tower

- Blue solid line drawn between UE and tower
- Updates continuously during dragging or mobility events
- Automatically disappears when UE leaves coverage

Throughput and Data-Rate Monitoring: the simulator tracks…

- Per-UE actual transmitted bytes
- Per UE maximum possible throughput based on PHY limits
- Per-tower aggregate throughput and internal buffer usage
- These metrics are displayed through the GUI and internal logs, supporting experiments involving congestion and ARQ behavior.

Automatic Handover Logic: UEs continuously evaluate tower distances and..

- Select the best available tower. When outages occur, UEs will automatically find a new and ideal tower.
- Switch bands automatically (high → mid → low)
- Trigger handovers with printed logs (e.g., “UE 103 handover from Tower 4 to Tower 7”)

Realistic IP addressing:

- Each UE and tower is assigned a unique 32-bit IPv4 address displayed in standard dotted-decimal format (e.g., 10.0.0.5) throughout the GUI
- IP addresses are allocated sequentially from a private address space to ensure uniqueness and consistency
- Data transmission is modeled using explicit IPv4 packets rather than abstract byte counts.
- Packets include realistic header fields such as source address, destination address, TTL, protocol identifier, and checksum.
- Payloads exceeding the IPv4 maximum transmission size are automatically fragmented
- Towers act as layer-3 routing nodes, delivering packets locally or forwarding them across tower-to-tower backhaul links.
- Hop counts are tracked per packet, and packets are dropped when a configurable limit is exceeded, preventing routing loops

Methodology

This simulator focuses on simulating real-world 5G scenarios, where noise and packet drop-outs may occur when transmitting data between devices (UEs). This simulator's main focus is the implementation of IPv4. Although we do not send real data in the GUI, functionality is in place to do so. As

is, the GUI sends random data to simulate network traffic throughout the system. If the user wishes to send real data (say through CLI), reference the following function in the UE class:

```
# Inputs are converted to byte arrays. The inputs must be lists.
# Header IDX's:
#    [ 0] - Version (Set to 4)           -  4-bits
#    [ 1] - Internet Header Length (IHL) -  4-bits
#    [ 2] - Type of Service (ToS)        -  8-bits
#    [ 3] - Total Length (header + data) - 16-bits
#    [ 4] - Identification (fragment)    - 16-bits
#    [ 5] - Flags                        -  3-bits
#    [ 6] - Fragment Offset              - 13-bits
#    [ 7] - Time to Live (TTL)           -  8-bits
#    [ 8] - Protocol (TCP=6, UDP=17)     -  8-bits
#    [ 9] - Header Checksum              - 16-bits
#    [10] - Source Address               - 32-bits
#    [11] - Destination Address          - 32-bits
#    [12] - Options                      - 0->40 bytes
def set_cust_data(self, header, data):
```

Code 1 - Send Real Data (Function)

The above send_cust_data() function allows the user to send custom or real data. The user must pack all the custom header fields into the header list. The header information / list items are then converted into a byte array which forms a packet. These *header* items are parsed and masked to follow the IPv4 protocol. The input *data* should also be a byte array, storing the data the user wishes to send. Again, with this function, the user can send *REAL* traffic throughout the network. Future development for this code will be to incorporate a segment that binds real traffic to the UEs. This would allow the user to stream and interact with real data through our simulated network.

Next, we would like to cover the implementation for each of the layers. For the *Physical* layer, we simulate channel effects through stochastic models. Meaning, we model random noise based on probability models, which affects SNR and BER. Next, we simulate 3 different usable frequency bands. The three bands are *High, Mid* and *Low* bands. Maximum coverage ranges include 300, 1500, and 5000 meters respectively. These bands also define a maximum data rate of each UE when occupying the respective band. The maximum data rates are as follows:

- 1000 Mbps for High band
- 200 Mbps for Mid band
- 50 Mbps for Low band

For the *Datalink* layer, we simulate connectivity via scanning between the towers and UEs. To connect to a cell, a UE should be in a connectable distance. If multiple cells are in range, the UE always selects the cell which provides the best throughput. If the cell which a UE is connected to drops out, the UE will scan and search for another cell to connect to. The UE periodically scans for new cells to connect to the best (highest throughput) link. In essence, we use a greedy method to connect UEs to cells. The cells use radial scanning to search for towers within connectable range, ensuring SNR is not too low. FEC (LDPC) is also implemented to simulate bit recovery. We base our FEC on LDPC waterfall curves which we found online [2]. Last, we implement ARQ to retransmit data after a set timeout period. This improves reliability of the network if packets get lost, dropped or corrupted during transmission.

For the *Network* layer, we again simulate IPv4 which is our highlight feature. All network routing is based on this topology. Initially we used shortest path routing (DFS then Dijkstra's algorithm), but we

found the simulator to run far too slow for real time calculations / updates when moving UEs. We chose to stick with a simple multi-casting method, utilizing the TTL field of the IPv4 header. This simplifies the network routing, but greatly improves simulation performance. We wish to improve on this routing algorithm in future iterations of this simulator.

Summary Of Assumptions & Abstractions

Simplified Physical layer:

The Physical layer is represented using SNR-based models derived from the relative positions of devices. This approach allows the model to estimate link quality without computing detailed radio propagation. It significantly reduces computational overhead, making large-scale simulations more feasible. Despite these simplifications, the model still captures meaningful environmental variation through SNR dynamics.

Simplified bit recovery:

Bit recovery relies on a stochastic process that uses SNR to estimate the probability of bit errors. This eliminates the need for implementing full modulation and coding schemes, which would add significant complexity. The stochastic approach still reflects realistic noise-related effects on transmission accuracy. It also offers flexibility for experimenting with different noise or interference conditions.

Simplified neighbor discovery:

Neighbor discovery is performed using radial scanning, identifying all nodes within a defined radius. Connection range is scaled based on the maximum sub-band range. The method is computationally efficient and scales well as node density increases. It also provides a predictable and easily tunable mechanism for neighborhood formation.

Model assumption on user intent:

The model assumes that users are primarily interested in simulating network traffic under custom, configurable scenarios. This enables broad experimentation with mobility patterns, traffic loads, and network topologies. Users can tailor the simulation environment to match research, instructional, or prototyping needs. The flexibility of this assumption supports both simple demonstrations and advanced analytical studies.

Installation & System Requirements

Python Requirements

The simulator requires:

- **Python : 3.9.13 and above**

Required Project Files

Ensure the following files remain in the same directory:

gui.py
tower.py
ue.py

transmission_test.py (for CLI only purposes)

User Manual

Starting the GUI simulator

Run the simulator using:

```
python gui.py
```

If successful, a window will appear showing:

- A hexagon canvas (main simulation area)
- Interactable control panel on top

Simulator Canvas

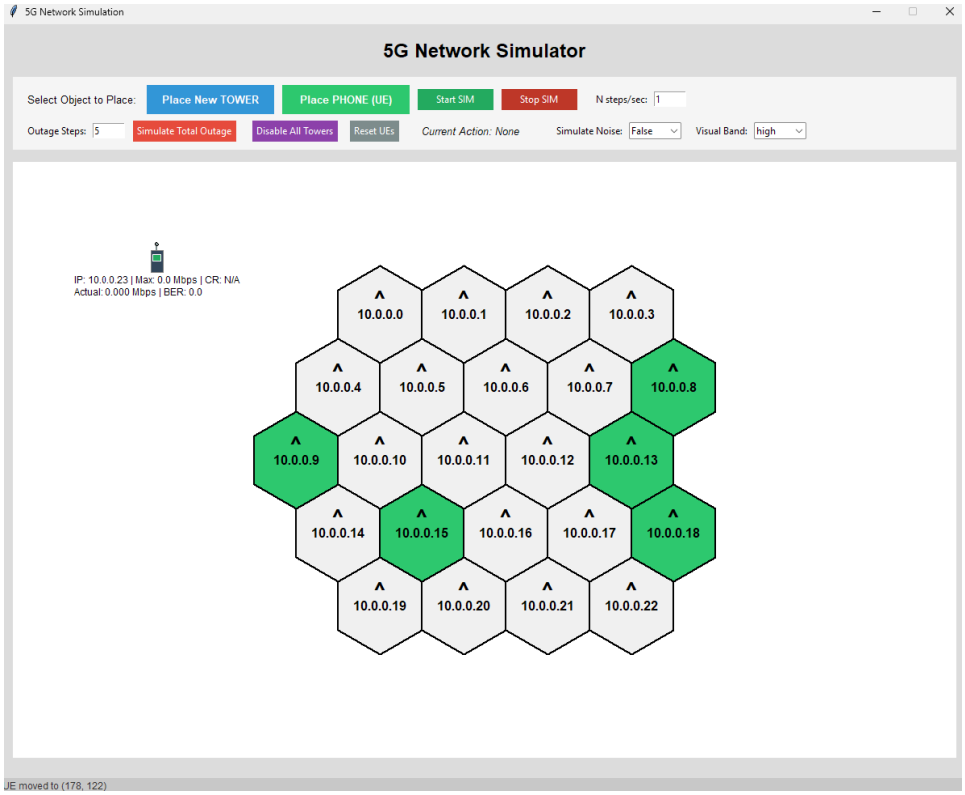


Figure 1 - Startup Screen

Overview of the User Interface

The canvas displays:

- Towers
- User Equipment (UEs)
- Coverage regions (high-band / mid-band / low-band)

- Tower–UE connection lines
- Throughput overlays

Control Panel

Located on the top side of the window:

Top Control Panel

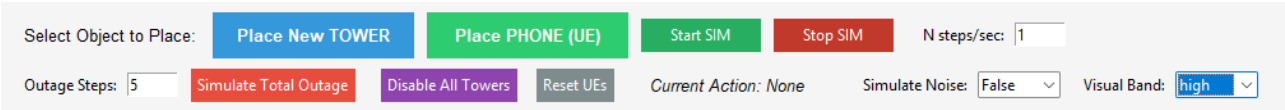


Figure 2 - UI Control Panel

- **Place New TOWER** – places a tower at the location you left-click, and snaps cleanly into the hexagon canvas
- **Place PHONE (UE)**– places a phone at the location you left-click. The phone can be dragged around after placement
- **Start SIM** – starts the simulation. Information can be seen in the terminal
- **Stop SIM** – pauses the SIM
- **Simulate Total Outage** – simulates all towers going down
- **Disable all Towers –** disables all towers
- **Reset UEs –** resets properties of UEs back to their defaults
- **Simulate noise –** toggles random Noise calculations on and off.
- **Visual Band –** allows the user to switch between bands, which affects the distance at which towers and UEs can connect. The hexagons are reflective of these bands.
- **N steps/sec –** allows the user to dictate how many ticks per second for the simulation. Every tick corresponds to Tx/Rx calculations between devices.
- **Outage Steps –** dictates how many steps the total outage lasts.

Device panels

Tower Properties Panel:

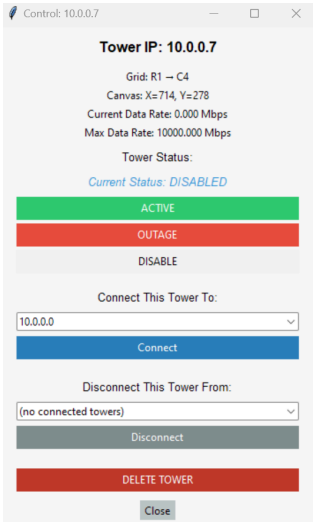


Figure 3 - Tower Properties

Open the tower's property panel: left-click tower(hexagon)

- The panel displays the tower's IP address, location, Current Data Rate, Max Data Rate, and current status.
- **Status** can be activated, disabled, or you can simulate an outage for that specific tower.
- **Connect This Tower To:** allows the tower to connect to another tower of choice
- **Disconnect This Tower From:** allows the user to disconnect a tower from the currently selected one.
- **DELETE TOWER:** will delete the entire tower from the canvas and its properties.

UE Properties Panel:
Open the selected UE's property panel: left-click the UE.

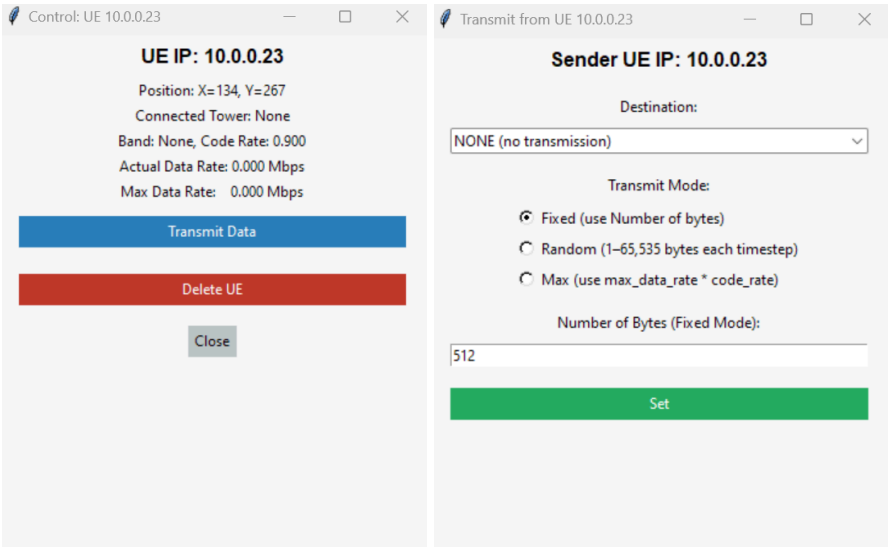


Figure 4 & 5 - UE Properties Panels

- Panel displays Position, connected tower, band, CodeRate, actual data rate, and Max Data rate.
- Transmit Data: allows you to choose which UE you want to transmit data to and how much.
- Delete UE: deletes UE from canvas

Simulation Display Options (Visual Band):

- Visual band HIGH/MID/LOW - allows the user to change the band in which the UEs can connect to the towers.

Command Line Interface

If the user wishes to use a non-GUI interface to control UEs, one can directly use the UE and Tower classes provided in the respective python files. First, the user must import the respective classes:

```
from ue import UE
from tower import Tower
```

Code 2 - Importing the Libraries

Once imported, the user must create a Tower's and UE's list. Towers and UEs share the same inputs:

- A device Identifier
- Starting x-position
- Starting y-position
- A list of connectable Towers
- A time delta (1 / n_ticks_per_sec)
- And an IPv4 Address

```
# Example of creating a towers list
towers = [Tower(id, x_pos, y_pos, t_delta, ip_addr) for id in range(n)]
# Example of creating a UEs list
ues = [UE(id, x_pos, y_pos, towers, t_delta, ip_addr) for i in range(m)]
```

Code 3 - Creating the Devices

Next, a simple simulation can be run. It is recommended to base your code on transmission_test.py, since there are multiple examples which simulate:

- Transmission of random data
- Random movement of UEs
- Tower shut-downs
- And simple telemetry examples

If the user wishes to run our simulation using Command Line Interface (CLI), all that needs to be done is to run transmission_test.py. Once run, some telemetry should print on screen. It is not as in-depth as the GUI telemetry, but it should serve great purpose when training the user on how to use our library. This transmission_test.py file was used in preliminary testing, before moving to the GUI. This is why it is a bit lacking in features:

```
Timestep 1478: Completed.
Tower IP_ADDR 0: Data rate = 0.022264 Mbps, Max data rate = 10000.0 Mbps
Tower IP_ADDR 1: Data rate = 0.046919999999999996 Mbps, Max data rate = 10000.0 Mbps
Tower IP_ADDR 2: Data rate = 0.022264 Mbps, Max data rate = 10000.0 Mbps
UE IP_ADDR 50: Tower IP_ADDR = 1 Band = mid, Code rate = 0.6666666666666666, Data rate = 0.007336 Mbps, Max data rate = 40.0 Mbps
UE IP_ADDR 51: Tower IP_ADDR = 1 Band = mid, Code rate = 0.6666666666666666, Data rate = 0.0 Mbps, Max data rate = 40.0 Mbps
UE IP_ADDR 52: Tower IP_ADDR = 1 Band = mid, Code rate = 0.6666666666666666, Data rate = 0.002224 Mbps, Max data rate = 40.0 Mbps
UE IP_ADDR 53: Tower IP_ADDR = 1 Band = mid, Code rate = 0.6666666666666666, Data rate = 0.009944 Mbps, Max data rate = 40.0 Mbps
UE IP_ADDR 54: Tower IP_ADDR = 1 Band = mid, Code rate = 0.6666666666666666, Data rate = 0.004984 Mbps, Max data rate = 40.0 Mbps
Timestep 1479: Completed.
Tower IP_ADDR 0: Data rate = 0.026327999999999997 Mbps, Max data rate = 10000.0 Mbps
Tower IP_ADDR 1: Data rate = 0.06036 Mbps, Max data rate = 10000.0 Mbps
Tower IP_ADDR 2: Data rate = 0.026327999999999997 Mbps, Max data rate = 10000.0 Mbps
UE IP_ADDR 50: Tower IP_ADDR = 1 Band = mid, Code rate = 0.6666666666666666, Data rate = 0.007336 Mbps, Max data rate = 40.0 Mbps
UE IP_ADDR 51: Tower IP_ADDR = 1 Band = mid, Code rate = 0.6666666666666666, Data rate = 0.007535999999999999 Mbps, Max data rate = 40.0 Mbps
UE IP_ADDR 52: Tower IP_ADDR = 1 Band = mid, Code rate = 0.6666666666666666, Data rate = 0.004064 Mbps, Max data rate = 40.0 Mbps
UE IP_ADDR 53: Tower IP_ADDR = 1 Band = mid, Code rate = 0.6666666666666666, Data rate = 0.009944 Mbps, Max data rate = 40.0 Mbps
UE IP_ADDR 54: Tower IP_ADDR = 1 Band = mid, Code rate = 0.6666666666666666, Data rate = 0.004984 Mbps, Max data rate = 40.0 Mbps
```

Figure 6 - Command Line Interface Telemetry

Again, we don't necessarily recommend using CLI unless GUI overheads are a concern. It is best to stick to using the GUI if a simple and intuitable interface is what the user wishes to work with.

```python
def simulation_main(towers, ues):
    while 1:
        # Make sure all timesteps are the same
        for ue in ues:
            ue.t_step = t_step

        # Example 1: Test transmitting from all UEs at once
        for ue in ues:
            dest_ip = ues[0].ip_addr # Or any random IP_ADDR
            ue.set_tx_bytes(n_bytes=random.randint(1, 1518), dest_ip=dest_ip)

        # Step through the calculations
        for ue in ues:
            ue.step()

        # Continually transmit from towers until
        # they reach capacity.
        can_tx = True
        tx_count = 0
        while can_tx:
            can_tx = False
            for tower in towers:
                if tower.can_transmit():
                    can_tx = True
                    tower.step()
                    tx_count += 1

        # Example 2: Printing data rates
        # Print actual data rate and max data rate of each device
        for tower in towers:
            print(f"Tower IP_ADDR {tower.ip_addr}: Data rate = {tower.n_tx_bytes * 8 * 1e-6} Mbps,
Max data rate = {tower.max_data_rate * 1e-6} Mbps")

        for ue in ues:
            if ue.current_tower is not None:
                print(f"UE IP_ADDR {ue.ip_addr}: Tower IP_ADDR = {ue.current_tower.ip_addr} Band =
{ue.freq_band}, Code rate = {ue.code_rate}, Data rate = {ue.n_tx_bytes * 8 * 1e-6} Mbps, Max data
rate = {ue.max_data_rate * 1e-6} Mbps")

        # Clear the transmission counter per timestep
        # Clearing for towers is NECESSARY!
        # this is so that we can run the while can_tx loop
        # until the towers reach the max data rate
        for tower in towers:
            tower.clear_tx_count()
        for ue in ues:
            ue.clear_tx_count()

        time.sleep(t_delta)
        print(f"Timestep {t_step}: Completed.")
        t_step += 1
```

Code 4 - CLI / Python Simulation Example

Experiments

Experiment 1

- **Send data between different UEs through the IP network.**

To run this experiment, ensure multiple UEs are on screen and click START SIM 

. If not enough UEs are on screen, click the button Place PHONE (UE) 

and click on the screen where you wish to place the device. Below is an example of two devices within the environment:

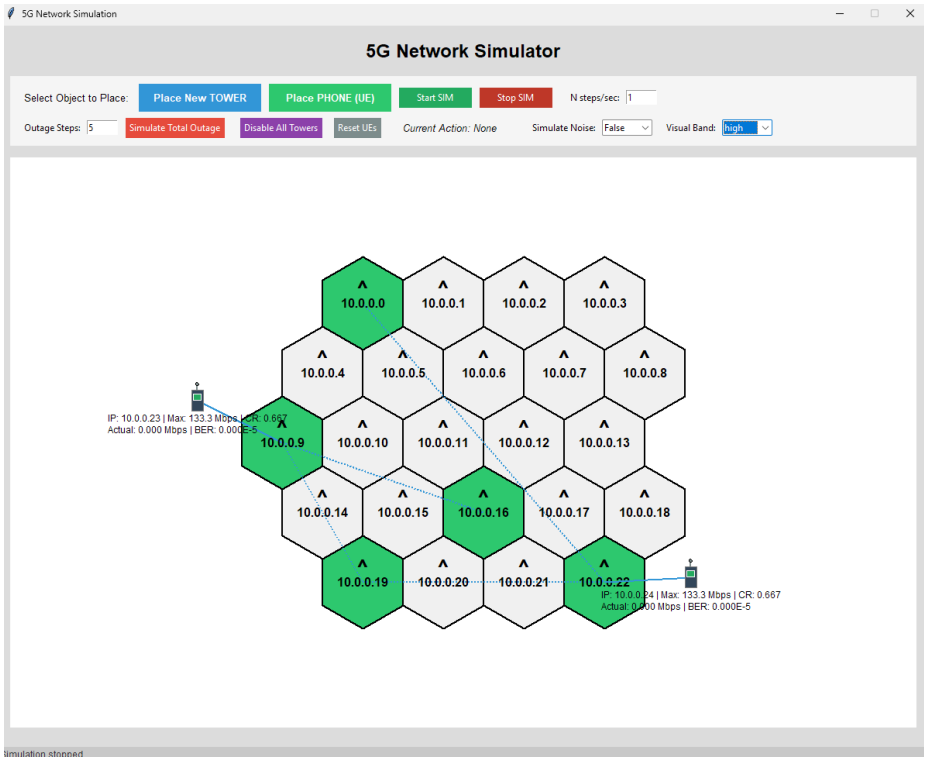


Figure 6 - Multi-device connectivity

Next, left-click on a given UE. This should bring up a dialog window with two options: Transmit Data and Delete UE. Transmit Data allows the user to transmit random data from device-to-device. The delete button completely deletes the UE from the environment. Click “Transmit Data”:

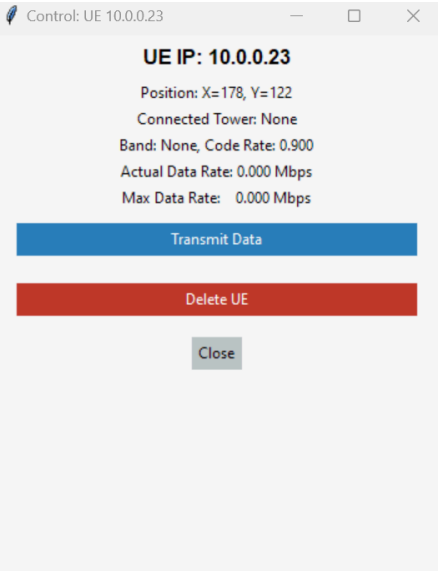


Figure 7 - UE Dialog Box

Next, another dialog box will appear. This box allows the user to select which destination address to send data to and the amount of data to send. If sending a fixed number of bytes, input your desired bytes to transmit in the box labeled “Number of Bytes (Fixed Mode).” It is recommended to use the “Random” selection for simplicity. Below are examples of the full window, along with the “Destination” drop down menu. It is recommended to select a single destination IP since broadcasting could cause the simulation to run slowly:

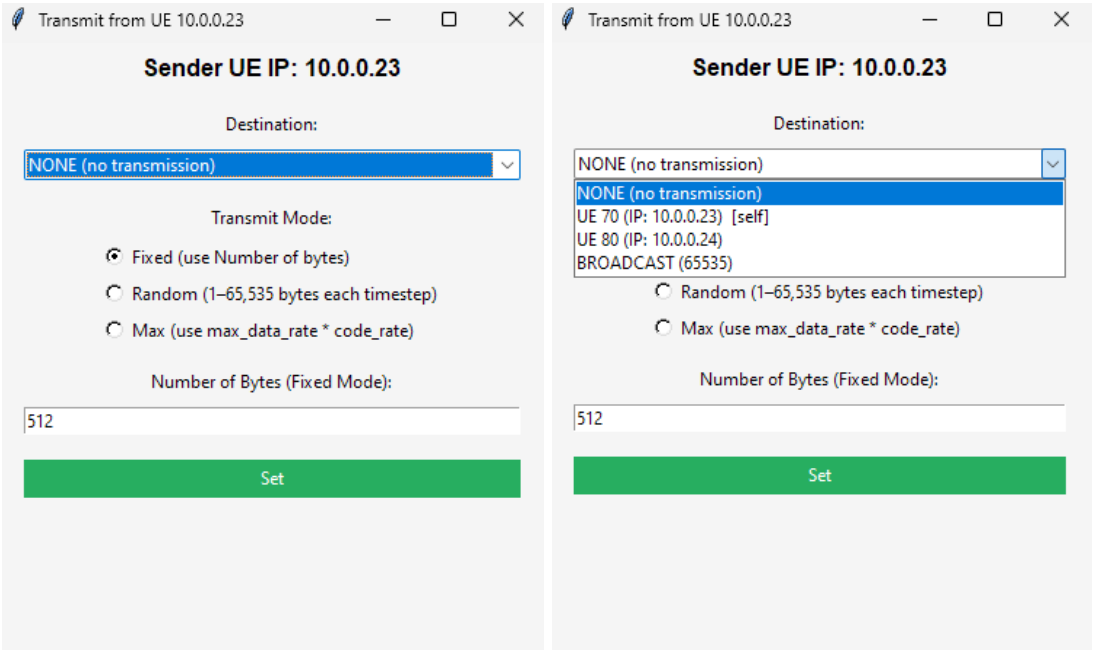


Figure 8 & 9: UE Transmit Data Menu & Destination Configuration

Once data transmission is configured you should see packet retrieval updates in your terminal. These represent the packets you configured to transmit in the GUI. The Command Line Interface (CLI) illustrates addressing, code rates, data rates, and bit-error rates per timestep:

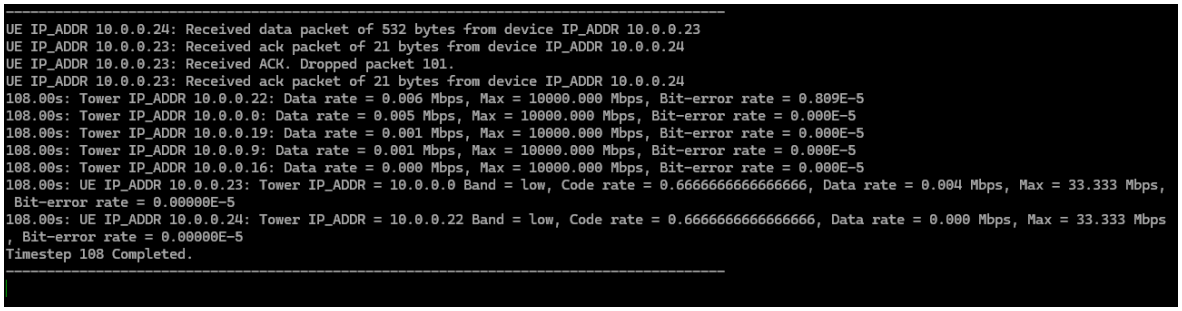


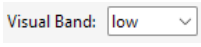
Figure 10 - Command Line Interface (CLI)

As the results show, we can successfully transmit data between devices via IPv4. All network traffic is buffered and routed through a dynamic network.

Experiment 2

- **Move an UE between different cells**

This one is straight forward, given our simple GUI. It is as simple as clicking a UE and dragging it. This experiment should be performed after experiment 1, where we begin transmission of data between UEs. All UEs are completely draggable, allowing users to dynamically configure their environment. UEs scan for the nearest tower to connect to. If a UE cannot connect to a tower, it will buffer any queued data. Using ARQ, the UE will retransmit any buffered data once connected to a new tower. As the user drags the UE across the environment, telemetry should update real-time, indicating data rates and such.

Different visualization modes can be set if you wish to adjust the radius of each hexagon or cell. In the visual band menu 

, the user can select different ranges for each cell. Each UE and Cell supports low, mid and high bands. Again, maximum coverage ranges include 300, 1500, and 5000 meters respectively. Once the Visual Band is set, the UE and Cell ranges will be adjusted real time in the GUI.

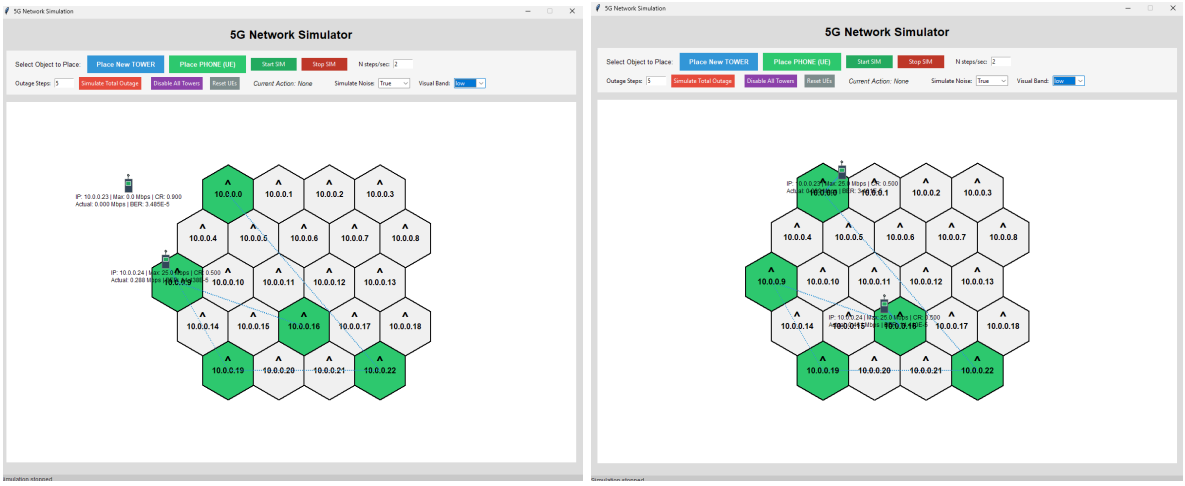


Figure 11 & 12 - Moving UEs (Click & Drag)

As shown above, we can successfully move UEs throughout the environment with ease. Telemetry is updated real time for user needs. Data will be buffered via ARQ if the UE goes out of range.

Experiment 3

- **Turn off a station tower and let other towers take over given the coverage.**

Another straightforward experiment is disabling random towers. Our GUI allows the user to easily disable a tower or simulate an outage. Given the grid below, say we want to simulate an outage on Cell IP_ADDR: 10.0.0.19. To do so, left click on said Cell. The following window will appear:

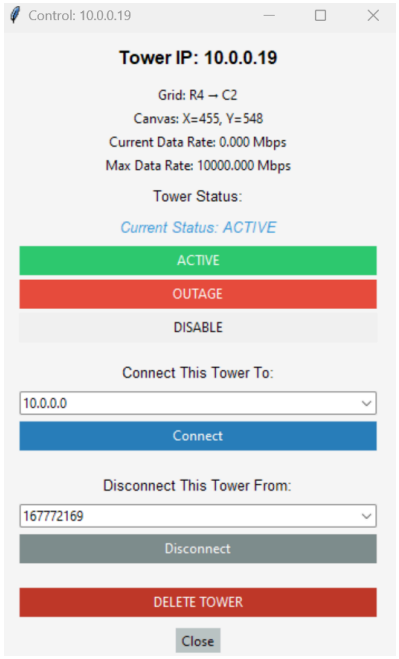


Figure 13 - Cell/Tower Configuration

Within this menu the only option the user needs to set is OUTAGE. This keeps the tower connections as is (the blue dotted lines on the grid), but forces the UEs to connect to a new device for transmission. The UE should re-connect to another Cell according to the outage:

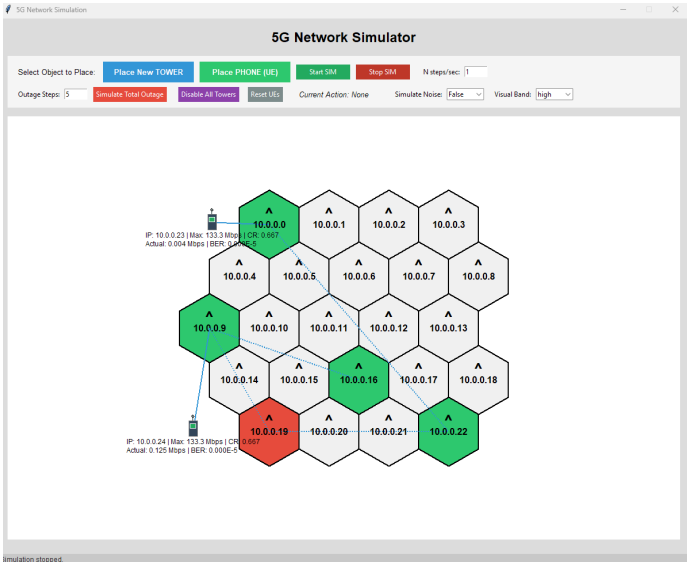


Figure 14 - Single Cell Outage

If the user wishes to re-enable the respective tower, left-click on the cell again and select ACTIVE. The tower should function as intended:

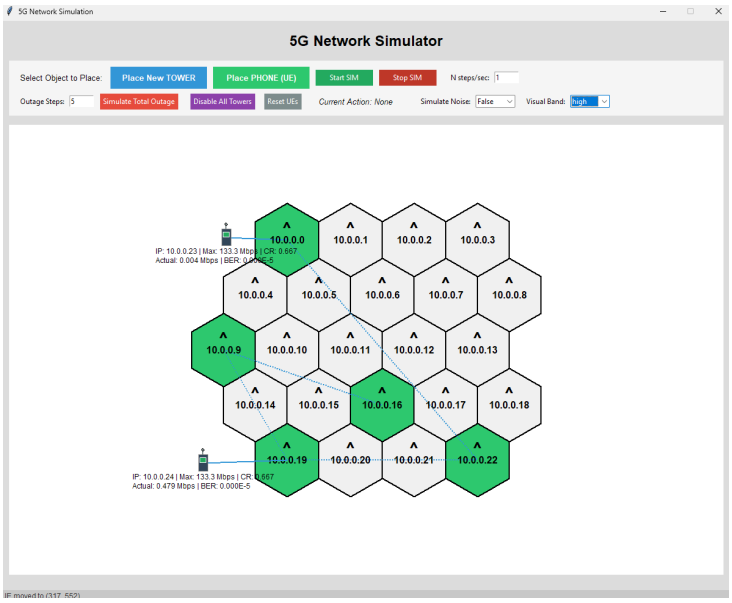
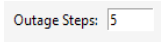


Figure 15 - Single Cell Outage Recovery

If the user wishes to simulate a total outage (all cells go down for a specified duration), we provide a simple clickable button. In the “Outage Steps” box 

, set the number of ticks you wish to experience the outage. Once set, it's as easy as clicking “Simulate Total Outage.” All towers should go red, indicating an outage. All UEs on the grid should disconnect from the towers during this outage. Once the specified number of ticks has passed, all Cells should turn green, indicating ACTIVE:

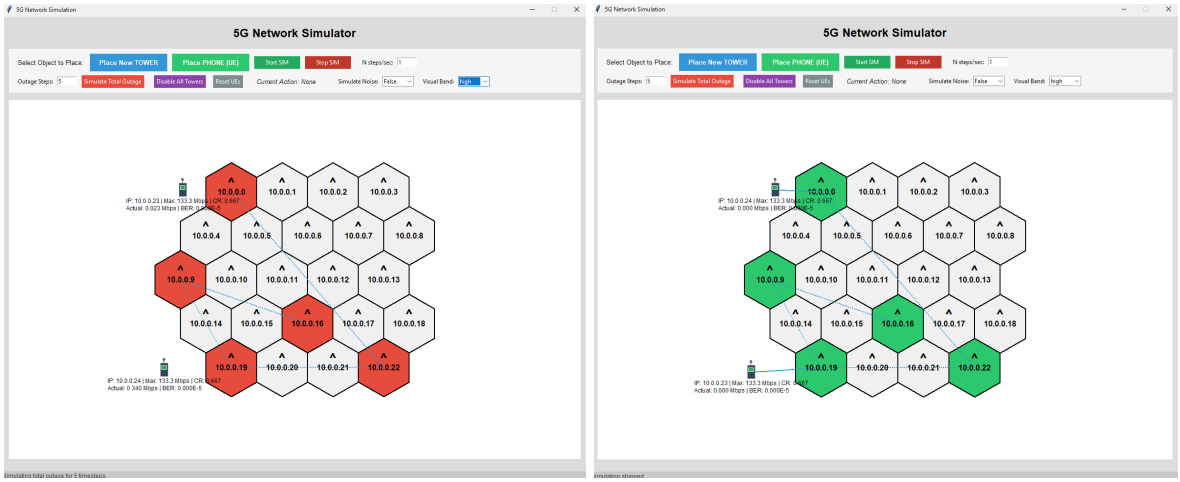


Figure 16 & 17 - Total Cell Outage and Recovery

As shown from our results, we are successfully able to disable Cells and allow UEs to re-discover and reconnect to new cells. In the case where no Cells are available to connect to, the UEs will buffer their data until an ARQ max re-transmission count is passed, or the UE can dump all its contents.

Future Improvements

A list of future improvements we would like add to the simulator is the following:

- Implementing different IP routing algorithms that are not as computationally expensive
- Add additional functionality to the GUI to include more stats and granular control over User Equipment and Tower Parameters
- Implement real data transmission for better network visualizations
- Rewrite code in a faster performing language to improve data transmission speeds
- Polish the GUI

Distribution of work

- Network Code - Manny, Russell, Anthony
- GUI Code - Manny, Santiago, Khang, Kyle, Diego
- Presentations - Manny, Russell, Anthony, Santiago, Khang, Kyle, Diego
- Report - Manny, Russell, Anthony, Santiago, Khang, Kyle, Diego

References

[1] 3GPP, “5G system overview.” https://www.3gpp.org/technologies/5g-system-overview (accessed Sept, 27, 2025).

[2] A. A. Author(s), “Comparison of 5G-NR LDPC and polar code over an AWGN channel employing GFDM modulation,” *ResearchGate*. https://www.researchgate.net/figure/Comparison-of-5G-NR-LDPC-and-polar-code-over-an-AWGN-channel-employing-GFDM-modulation_fig4_347771673 (accessed Sept, 27, 2025.
(Note: ResearchGate items often lack full citation metadata; replace Author(s)/title with the actual paper details if known.)

[3] Python Software Foundation, “tkinter — Python interface to Tcl/Tk,” *Python Docs*. https://docs.python.org/3/library/tk.html (accessed Sept, 27, 2025).

[4] SimPy Developers, “SimPy documentation.” https://simpy.readthedocs.io/en/latest/ (accessed Sept, 27, 2025).

[5] A. S. Tanenbaum and D. J. Wetherall, *Computer Networks*, 5th ed. Boston, MA, USA: Pearson, 2011.

[6] Wikipedia, “IPv4.” https://en.wikipedia.org/wiki/IPv4 (accessed Sept, 27, 2025).