

# NetSimulyzer: a 3D Network Simulation Analyzer for ns-3

Evan Black

National Institute of Standards and Technology  
Gaithersburg, Maryland, USA  
evan.black@nist.gov

Samantha Gamboa

Associate, National Institute of  
Standards and Technology  
Prometheus Computing LLC  
Sylva, North Carolina, USA  
samantha.gamboa@nist.gov

Richard Rouil

National Institute of Standards and Technology  
Gaithersburg, Maryland, USA  
richard.rouil@nist.gov

## ABSTRACT

The increased complexity of network protocols and scenarios simulated using ns-3 is making the verification of simulation correctness and the analysis of simulation outputs a challenging task. In this paper, we present a new and flexible visualization tool for ns-3, called NetSimulyzer, that can alleviate the workload of debugging, understanding, and demonstrating a scenario. The tool was conceived to easily integrate to any ns-3 scenario and provides core functionalities that are technology agnostic. NetSimulyzer provides mechanisms to track a variety of simulation elements, from topology and node mobility, to statistics and other data generated by the simulated network protocols. The collected information can be visualized using a 3D scene augmented with data visualization elements such as charts and logs. In this paper, we provide an overview of the architecture and functionalities of the tool, and we also illustrate its usability and versatility by visualizing scenarios provided in the standard ns-3 distribution.

## CCS CONCEPTS

• **Networks** → **Network simulations**; • **Human-centered computing** → **Visualization systems and tools**.

## KEYWORDS

ns-3, network visualization, 3D visualization, Qt

### ACM Reference Format:

Evan Black, Samantha Gamboa, and Richard Rouil. 2021. NetSimulyzer: a 3D Network Simulation Analyzer for ns-3. In *2021 Workshop on ns-3 (WNS3 2021)*, June 23–24, 2021, Virtual Event, USA. ACM, New York, NY, USA, 8 pages. <https://doi.org/10.1145/3460797.3460806>

**Disclaimer:** Any mention of commercial products in this paper is for information only; it does not imply recommendation or endorsement by NIST.

## 1 INTRODUCTION

The ns-3 Network Simulator is a discrete-event simulator widely used in research and academia [9]. It can simulate a variety of communication network architectures and protocols. As open-source software, users can also extend or implement new models that can use the underlying simulation framework. A typical workflow

when using ns-3 includes the creation and parameterization of a scenario, the execution of the simulation(s), and the analysis of the simulation output data. Users can enable the ns-3 logging system or use the standard outputs to monitor the progress of the simulation. In addition, users make use of the tracing system, the modules' built-in metric systems, or the data collection framework [10], to gather simulation data of interest.

Interpreting network performance has become more complex due to several factors, including the increasing number of devices, the complexity of the protocols, the integration of different technologies, and the deep connection between upper and lower layer performance. Thus, relying upon raw data outputs to debug, understand, and demonstrate a scenario is cumbersome. Users often rely on custom post-processing tools that are scenario and technology specific and difficult to generalize or scale.

Separate visualization software may be used to ease simulation interpretation. Such software allows ns-3 users to reproduce the simulation and display performance metrics in a user-friendly manner. Visualizers also facilitate verifying correct scenario implementation such as node topology, simulation environment, and event timelines, and allow users to study protocol behavior through state machines and message exchanges.

There are two visualizers included with ns-3: NetAnim [4] and PyViz [11]. NetAnim is an offline visualizer, i.e., it uses an eXtensible Markup Language (XML) trace file generated at the end of a simulation to replay events in a separate application. PyViz is an online visualizer, i.e., it runs together with the simulation allowing live representation and debugging of a scenario. Both visualizers can show 2D representations of the scenario topology and animate node mobility and network connections between nodes. NetAnim can animate packet exchange between nodes over the displayed connections, while PyViz can display link statistics (e.g., data rate). NetAnim software integrates some summary tabs that can show metadata captured during the simulation, e.g., packet flows, node counters, and routing tables.

While NetAnim and PyViz are very useful for visualizations that only require the above features, extending the information to trace and display data or metrics, other than the ones embedded in their corresponding modules, is not trivial. In previous work, we built a demonstration of Mission-Critical Push-To-Talk (MCPTT) protocol capabilities over Long Term Evolution (LTE) Device-to-Device (D2D) communications using PyViz [6]. In addition to the limited documentation, which delayed our assessment of the tool, the tight coupling between the visualizer and the ns-3 modules (e.g., python bindings) required a considerable amount of effort and programming expertise to augment PyViz and obtain satisfactory results. Another limitation of NetAnim and PyViz is the lack of support

ACM acknowledges that this contribution was authored or co-authored by an employee, contractor, or affiliate of the United States government. As such, the United States government retains a nonexclusive, royalty-free right to publish or reproduce this article, or to allow others to do so, for government purposes only.

WNS3 2021, June 23–24, 2021, Virtual Event, USA

© 2021 Association for Computing Machinery.

ACM ISBN 978-1-4503-9034-7/21/06...\$15.00

<https://doi.org/10.1145/3460797.3460806>

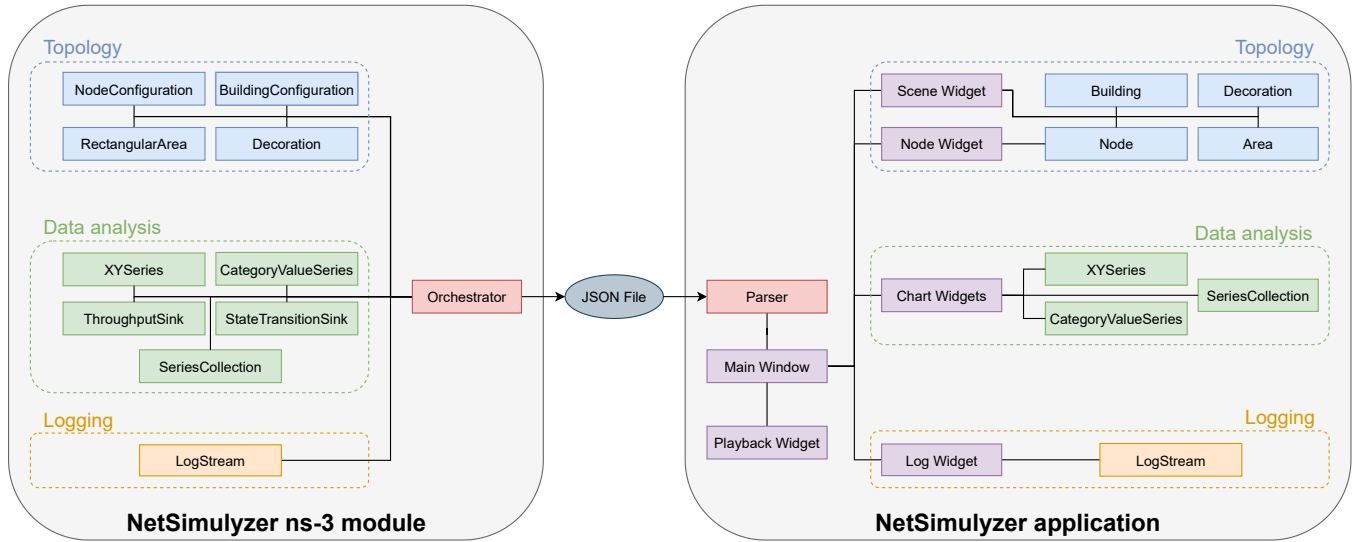


Figure 1: NetSimulyzer Architecture Overview

for 3D visualization even though ns-3 supports node positioning and mobility in 3D space. This likely prevents their effective use to support active research in aerial networks [2, 5, 12, 15] and millimeter wave communication [1, 18] where altitude and transmission angles play an important part.

Other users in the ns-3 community have implemented custom visualizers to suit their studies. For example, the authors in [17] describe a visualizer developed for the Institute of Electrical and Electronics Engineers (IEEE) 802.11ah ns-3 model. It is an online visualizer that can display a 2D representation of the topology, configuration, traffic statistics, and metrics of interest for 802.11ah (e.g., the slot usage) during simulation. In [1], the authors developed an offline visualizer to analyze beamforming training for the IEEE 802.11ad/ay ns-3 models using millimeter wave bands. This tool represents in 3D the scenario topology and environment, and the evolution of the beam steering configuration of the transmitters and receivers in the simulation. While these tools provide enhanced scenario visualization, they are currently limited to the specific technologies that the respective authors are studying. One open-source tool that aims to be generic is the United States Naval Research Labs (NRL)’s Scripted Display Tool (SDT) [16]. There are two versions of the tool: SDT, which provides 2D visualization like NetAnim and PyViz, and SDT3D, a Java-based application that provides 3D visualization by overlaying topology information on top of a map of the world. While it is possible to export an SDT3D compliant file from ns-3, it requires to use geographic coordinates to place the nodes on a map. Geographic coordinates are supported in ns-3 but not widely used. Furthermore, the terrain shown in the visualization is unlikely to represent what was modeled in the simulation unless the scenario is configured to use a terrain-aware propagation model such as Terrain Integrated Rough Earth Model (TIREM) available via contributed model to ns-3 [13]. Finally, SDT3D does not provide plotting capabilities to show network performance, thus requiring manual scripts to generate plots or use another library.

Due to the aforementioned limitations, we decided to implement a new visualizer that provides a 3D representation of the topology, core capabilities that are technology agnostic, and flexible simulation data collection and visualizations. Since the tool is meant to be used by other ns-3 users, it was also critical to make integration with existing and new scenarios as easy as possible.

The rest of the paper is organized as follows. Section 2 describes the structure and components of our tool and the functionalities they provide. In Section 3, we use several examples to demonstrate the capabilities and usability of the tool. Section 4 describes new functionalities that we are currently developing or plan to develop in the near future. Finally, Section 5 provides some concluding remarks.

## 2 NETSIMULYZER OVERVIEW

The NetSimulyzer is a tool designed to visualize and aid in understanding ns-3 scenarios of any size and any communication network technology used. NetSimulyzer is currently based on the offline approach and, as shown in Figure 1, it works by combining two separate entities: the NetSimulyzer ns-3 module, which reads the simulation data, and the NetSimulyzer application, which displays it. Both entities are open source and respectively available at [7] and [8].

The NetSimulyzer ns-3 module is used when configuring an ns-3 scenario and provides simple functions to specify the data to be collected during the simulation. The ns-3 module will generate a JavaScript Object Notation (JSON) file, which is used by the NetSimulyzer application to visually represent the simulation using different elements: a 3D scene that represents and animates simulation components in 3D, and multiple mechanisms to show the collected data in meaningful ways, e.g., with logs and interactive charts.

## 2.1 NetSimulyzer ns-3 Module

The NetSimulyzer ns-3 module is used to configure, collect, and export simulation information for display in the NetSimulyzer application. The module is designed to be modular, and the Orchestrator component is the only mandatory component that needs to be added to a scenario in order to use the tool. Every other component can be added based on what is relevant to the given scenario. Another design goal was the simplicity of integration; as such, the components typically require very few lines of code to add to existing ns-3 scenarios as shown in Section 3. In this section, we group the components currently present on the module by functionality and we describe their main characteristics and functions.

**2.1.1 Orchestrator.** The Orchestrator class is the base of the NetSimulyzer ns-3 module. It sets simulation-wide options, defines the output file path, and controls the output of tracked information to the JSON file. Every class of the module requires some reference to an Orchestrator, typically in the constructor. A single simulation may support several Orchestrator instances, allowing for several outputs from one simulation.

**2.1.2 Topology.** The module provides mechanisms to track the evolution of the simulation topology, supporting both items already found in ns-3, such as nodes and buildings, as well as additional items to further describe or enhance the environment and topology visualization, such as decorations and highlighted areas.

The class NodeConfiguration holds the visualization configuration of a node, such as its name and 3D model to use for display in the application, and monitors its position throughout the simulation. To begin tracking a node object for display, users aggregate a NodeConfiguration object onto it, and the module will automatically track the node for the rest of the simulation. The module will also automatically integrate with any of the ns-3 mobility models to easily track and output the node's location without additional code. The NodeConfiguration class supports many additional, optional properties to further refine the rendering of a node, such as configurable colors, height, orientation, offset, etc.

The NetSimulyzer ns-3 module is also capable of displaying buildings that are defined in the ns-3 scenario using a similar Application Programming Interface (API) to nodes, provided by the class BuildingConfiguration. To track a building object, users aggregate a BuildingConfiguration object onto it. Each building location, dimensions, and rooms are exported and will be displayed in the application as a solid or semi-transparent rectangular prism with planes dividing the rooms. The NodeConfigurationHelper and BuildingConfigurationHelper classes are also provided to allow simple configuration of many nodes and buildings at once.

To draw attention to locations of some significance in the simulation; the module provides the RectangularArea class. These areas are defined by an ns-3 rectangle and are displayed in the application with a border, fill, or both. The colors of the fill and border of an area can be configured in the scenario. Finally, to add purely visual enhancements to the topology scene, the module provides the Decoration class with a similar API to NodeConfiguration. This class alleviates the need to create nodes for props based on 3D models.

**2.1.3 Data Analysis.** Most of the network protocol information from an ns-3 simulation cannot be represented by the topology but instead must be captured via standard outputs or the ns-3 tracing system (i.e., by using trace sources and sinks).

The NetSimulyzer ns-3 module currently provides three generic classes to collect such information from the simulations and to visualize it in the application using plots. The XYSeries class tracks information that can be expressed using two numeric coordinates, e.g., metrics over time, 2D positions, etc. By default, the module produces a plot per XYSeries object, and additionally, the SeriesCollection class can be used to group several XYSeries objects to be displayed in a single plot. The CategoryValueSeries class tracks numeric data organized into discrete, String categories, e.g., the state names of a state machine. Each class supports a number of properties that may be configured via the ns-3 attributes system or functions for configuring the plots. These properties include plot title, axis label, colors for the series, and the type of plot (lines, dots, connected dots). After an XYSeries or CategoryValueSeries is created, the data can be appended to it during simulation time. While there is no dependency on the ns-3 tracing system to generate statistics for the NetSimulyzer, this is typically used when instrumenting a scenario. For example, using a XYSeries object for collecting information, as shown later in the example of Section 3.1.

The module also provides two helper sinks to collect and process data to calculate throughput and display state machine changes in an easy way. The ThroughputSink tracks total data written by a model over a configurable period of time, calculates the throughput, and uses an XYSeries object to produce a plot that shows the throughput over time. The user can configure the interval at which the throughput is calculated as well as the unit for the Y-axis (e.g., bit/s, MB/s) based on expected throughput. This sink was designed to easily connect to packet transmission (Tx) and reception (Rx) trace sources defined in most applications. The StateTransitionSink was designed to track state changes and uses a CategoryValueSeries object to plot the changes against the time they occurred. This sink works with models that provide a traced value for their current state that uses states stored as Strings or as Enumerated types.

**2.1.4 Logging.** The LogStream class provides a mechanism, independent of ns-3's logging framework (NS\_LOG), to output String messages during the simulation playback in the application. The LogStream class provides an API similar to `std::cout` and can be useful for displaying messages about the status of the simulation, such as marking the beginning of important events or indicating some failure condition has occurred. Each LogStream can be configured with a name and font color so that they can be clearly identified in the NetSimulyzer application during playback.

## 2.2 NetSimulyzer Application

The NetSimulyzer application is a lightweight, standalone, cross-platform, open-source, Qt application that replays an ns-3 simulation tracked by the NetSimulyzer ns-3 module. The application is a collection of optional widgets centered around the SceneWidget. The SceneWidget uses OpenGL to render hardware-accelerated 3D graphics for displaying the environment and network topology. The other optional widgets allow for control of the reproduction



Figure 2: Example of the NetSimulyzer Application Interface

of the simulation and the display of the collected data using charts and logs.

In this section, we describe the main components of the NetSimulyzer application grouped by functionality so that the reader can see the correspondence with Figure 1 and Section 2.1. Also, we present in Figure 2 an example of the NetSimulyzer application reproducing a scenario. In the figure, we also show a breakdown of the interface components that will be referred throughout this section.

**2.2.1 Parser.** The parser reads the output file from the NetSimulyzer ns-3 module into models understood by the NetSimulyzer application. It also collects some metadata about the simulation, such as the furthest points in each direction and the last event time to determine the ground plane size and how long to make the playback timeline.

**2.2.2 Main Window.** The MainWindow holds all of the widgets and provides areas on the top, bottom, left, and right to move these widgets, referred to as “docks”. Any widget, aside from the SceneWidget, may also be detached from the window and moved around freely.

**2.2.3 Playback.** Playback of the simulation can be controlled by the PlaybackWidget (item 1 in Figure 2). It allows for starting and stopping playback with either the play/pause button or pressing the P key. The user may also seek within the simulation by dragging the slider in the center of the widget forwards or backward. When the slider is moved, all the widgets are adjusted to the state that they were for the new time.

**2.2.4 Topology.** The traced topology from ns-3 is displayed in the widget at the center of the window, referred to as the SceneWidget (item 2 in Figure 2). It displays the nodes, buildings, decorations, and areas defined in the simulation and a ground plane, skybox, and minor lighting effects. The user may move or rotate the 3D scene using the mouse and keyboard. It is the only widget that may not be detached, moved, or disabled, although other widgets may be resized to cover it entirely.

The NodeWidget (item 3 in Figure 2) displays the list of nodes rendered by the application, with their node ID from ns-3, and the name set in the NodeConfiguration. The user may reorder the table by clicking on the column headings, and may change the view in the SceneWidget to center on a node by double-clicking on a record in the table.

**2.2.5 Data Analysis.** The application renders XYSeries, CategoryValueSeries, and SeriesCollection objects from the ns-3 simulation into plots. These plots may be displayed on a ChartWidget (item 4 in Figure 2), added to the window by opening the “Window” menu and clicking “Add Chart.” Several ChartWidget instances may be created to compare data between plots easily. The user may adjust the view of the chart and focus on specific areas by clicking and dragging with the mouse or by adjusting the zoom level using the keyboard.

**2.2.6 Logging.** The LogWidget (item 5 in Figure 2) manages the LogStreams from the NetSimulyzer ns-3 module. Each individual LogStream may be selected by name from the drop-down at the top of the widget. There is also a “Unified Log”, which displays messages from every LogStream with the name from the ns-3 module appended in front of each message.

### 3 EXAMPLES

In this section, we use several examples provided in the standard ns-3 distribution to showcase the capabilities of the visualizer. These scenarios and their respective JSON output files are provided with the visualizer to assist new users as they familiarize themselves with integrating the NetSimulyzer module with ns-3 scenarios, and so that it is not necessary to run ns-3 simulations prior to operating the tool. In addition, a recorded presentation describing and walking through the scenario shown in Figure 2 is also available in [14].

#### 3.1 Outdoor Random Walk

The first example demonstrates how easy it is to integrate the visualizer to an existing scenario to display the topology, add a plot, and use the log feature. To that extent, we selected the outdoor-random-walk-example scenario provided in the buildings module as it includes many buildings and one moving node. The code necessary to render Figure 3 is shown in Listing 1.

One powerful feature of the visualizer, which is to display the various buildings and moving nodes in the 3D scene widget, can be achieved by simply adding lines 15 to 23. We also show how to create and configure a log component and an XYSeries plot (lines 24 to 32). For the purpose of demonstration, both elements are used to export node position information, which occurs in the callback function defined at line 5. The user can then better understand the scenario layout by navigating through the topology and follow the node's movement after clicking the playback button. The resulting plot ends up representing a 2D trace of the node's trajectory, a capability also present in NetAnim.

#### 3.2 WiFi Bianchi

We now present an example based on the wifi-bianchi scenario provided in the Wi-Fi module that was developed to validate the performance of the Wi-Fi ns-3 model with the theoretical model developed by Bianchi [3]. In this scenario, a number of devices in very close proximity communicate with each other over Wi-Fi. The scenario supports many parameters, including the Wi-Fi technology used (e.g., 802.11b, 802.11g), packet size, duration, and connection mode (i.e., infrastructure or ad-hoc). The scenario is also logging various information across different layers, such as when devices start to transmit or receive a packet at the physical layer, medium access control (MAC) layer, and application layer, or when devices need to execute the backoff procedures.

In order to study the effect of congestion and enhance the analysis, the distance between the center device and the other devices is set to 10 cm and the traffic start of each device is staggered throughout the first 75 % of the simulation time. A screenshot of the visualization is shown in Figure 4, which can be obtained with about 200 lines of code (the scenario itself being over 1200 lines of code). On the left side, the list of nodes is displayed, which for this example shows 10 stations. The log shows the time at which each station starts sending traffic. The charts selected show plots of the total MAC throughput received across all the devices, the traffic between station 7 (blue phone) and station 8 (green phone), and the backoff duration at station 7. The combo boxes associated with each figure could be used to look at traffic between any pair

of nodes or view the congestion window information which is also available.

By looking at the various plots, the NetSimulyzer can be an effective tool to help students and researchers understand the impact of congestion in a Wi-Fi network. At time 8.89 s, station 7 starts to transmit. We can observe the increase in throughput in the top two figures, with station 7 sending at a rate of 12 Mbit/s (blue line in the second plot). Initially, all the traffic is received by node 8 (green line in the second plot). After the third station starts transmitting traffic, at time 34.52 s, the network saturates. Given that all the stations in the scenario have the same sensing configuration, they will each share the capacity equally. As such we see a gradual degradation of the traffic received by station 8, until about 3 Mbit/s, roughly 1/10 of the total capacity (as shown on the top figure). The bottom plot also shows the backoff time experienced by node 7. As more stations transmit, the medium gets busy for longer periods of time and stations have to backoff for longer periods of time.

#### 3.3 LTE Radio Link Failure

The final example, shown in Figure 5, is based on the lena-radio-link-failure scenario available in the LTE ns-3 module. The scenario is designed to test the radio link failure and handover capabilities by having a User Equipment (UE) moving at a constant speed across the coverage areas of various eNodeBs. Inputs to the scenario include the number of eNodeBs, counters for the radio link failures, control of error models, and simulation time. The outputs are used mainly to capture the state of the Radio Resource Control (RRC) connection at the UEs and eNodeBs.

During the integration with the NetSimulyzer, three LogStreams were defined: one for the applications, one for events happening at the eNodeBs, and one for events at the UEs. On the figure, we can see that different colors are assigned to each log, making it easy to read. Plots were created for the RRC state, aggregated received throughput (i.e. combining both uplink and downlink), and the signal strength of the surrounding eNodeBs measured by the Reference Signal Received Power (RSRP). Figure 5 also demonstrates the NetSimulyzer's ability to rearrange the layout, with the RRC state plot placed on the left of the 3D layout since it requires more space.

The selected plots show that the UE initially connects to cell 1 and the aggregated throughput is about 9 Mbit/s. As the UE moves away from cell 1, the signal strength becomes weaker, as shown by the green line in the RSRP plot. Consequently, the throughput decreases as the eNodeB performs link adaptation and reduces the Modulation and Coding Scheme (MCS) used for the UE. At 14.47 s, a radio link failure occurs, triggering a scan and a handover to cell 2. The RRC state diagram shows a transition to the CONNECTED\_PHY\_PROBLEM state before conducting a cell search and then re-establishing a connection. After the handover takes place, the UE is able to successfully send and receive packets, with increasing throughput as the UE gets closer to cell 2.

### 4 FUTURE WORK

As shown in the previous sections, the NetSimulyzer supports a large set of features to visualize and analyze ns-3 simulations. In this section, we introduce some new features currently in development or in our roadmap.

```
1 #include <ns3/netsimulyzer-module.h>
2 // ...
3 //Define callback function to track node mobility
4 void
5 CourseChanged (Ptr<netsimulyzer::XYSeries> posSeries, Ptr<netsimulyzer::LogStream> eventLog, std::string context, Ptr<const
    MobilityModel> model)
6 {
7     const auto position = model->GetPosition ();
8     //Write coordinates to log
9     *eventLog << Simulator::Now ().GetSeconds () << " Course Change Position: ["
10         << position.x << ", " << position.y << ", " << position.z << "]\n";
11     //Add data point to XYSeries
12     posSeries->Append (position.x, position.y);
13 }
14 // ...
15 auto orchestrator = CreateObject<netsimulyzer::Orchestrator> ("output.json");
16 //Use helper to define model for visualizing nodes and aggregate to Node object
17 netsimulyzer::NodeConfigurationHelper nodeHelper{orchestrator};
18 nodeHelper.Set ("Model", StringValue ("models/land_drone.obj"));
19 nodeHelper.Install (nodes);
20 //Use helper to configure buildings and export them
21 netsimulyzer::BuildingConfigurationHelper buildingHelper{orchestrator};
22 for (auto building = buildingVector.begin(); building != buildingVector.end(); building++)
23     buildingHelper.Install(*building);
24 //Create a LogStream to output mobility events
25 Ptr<netsimulyzer::LogStream> eventLog = CreateObject<netsimulyzer::LogStream> (orchestrator);
26 eventLog->SetAttribute ("Name", StringValue ("Event Log"));
27 //Create XYSeries that will be used to display mobility (like 2D plot)
28 Ptr<netsimulyzer::XYSeries> posSeries = CreateObject<netsimulyzer::XYSeries>(orchestrator);
29 posSeries->SetAttribute ("Name", StringValue ("Node position "));
30 posSeries->SetAttribute ("LabelMode", StringValue ("Hidden"));
31 posSeries->SetAttribute ("Color", netsimulyzer::BLUE_VALUE);
32 posSeries->GetXAxis ()->SetAttribute ("Name", StringValue ("X position (m)"));
33 posSeries->GetYAxis ()->SetAttribute ("Name", StringValue ("Y position (m)"));
34 //Tie together the callback function, LogStream, and XYSeries
35 Config::Connect ("/NodeList/*/ $ns3::MobilityModel/CourseChange", MakeBoundCallback (&CourseChanged, posSeries, eventLog));
```

Listing 1: Code Needed to Visualize outdoor-random-walk-example as Shown in Figure 3

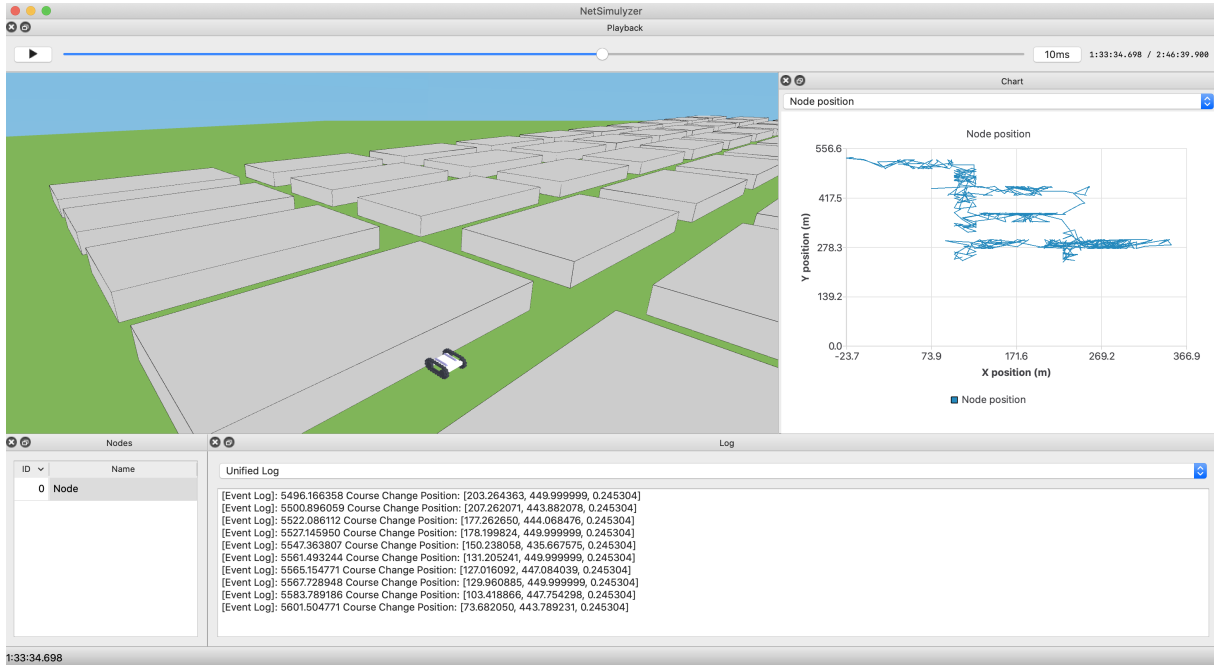


Figure 3: Visualization of outdoor-random-walk-example Scenario



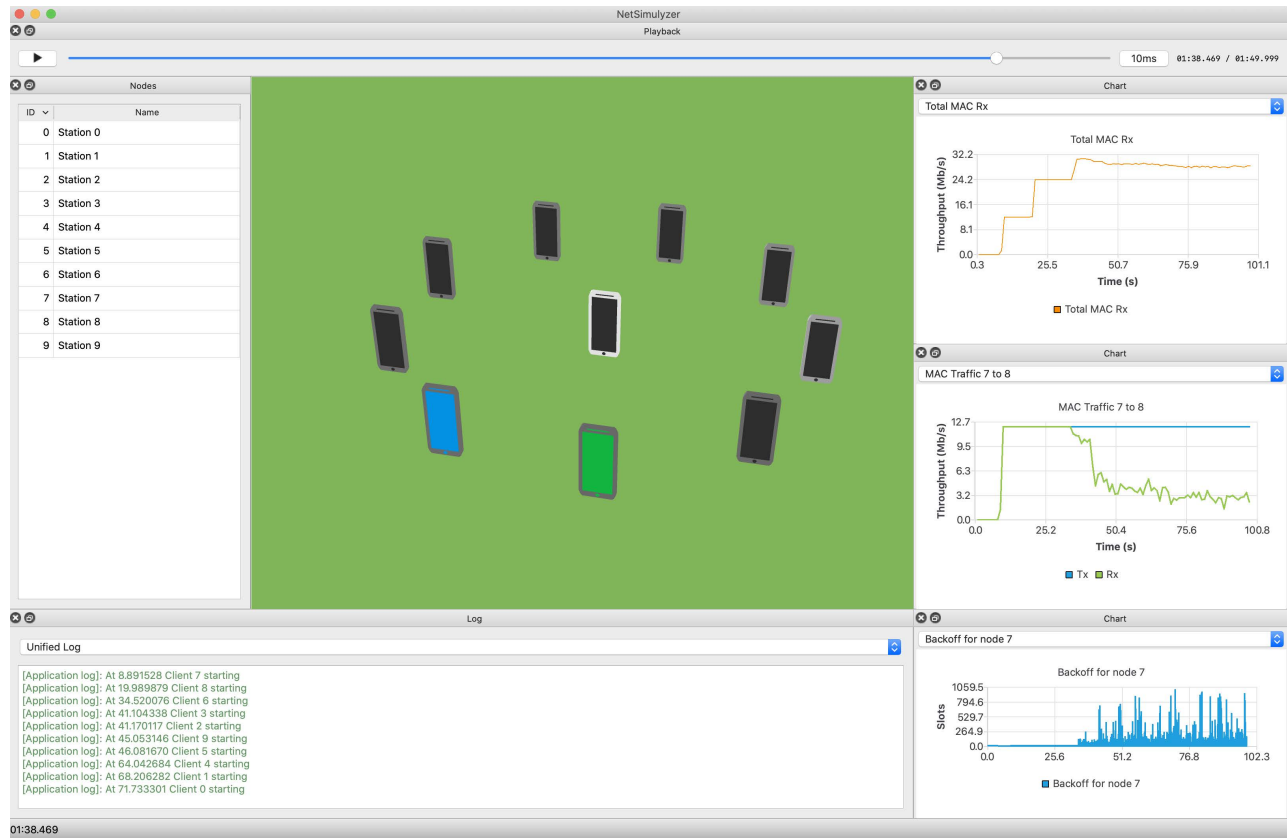


Figure 4: Visualization of wifi-bianchi Scenario

We are currently developing additional widgets to display per-node network information, such as a list of Applications and NetDevices, and their associated information (e.g., device type, IP, and MAC addresses). The goal is to provide dynamic node status information relevant to the user.

We plan to expand the statistics capabilities. This will be done by providing additional trace sinks in the ns-3 module to include other frequently traced metrics, such as latency. Additional chart types, such as histograms and Cumulative Distribution Function (CDF) plots will also be developed.

We also plan to enhance the 3D scene by providing more network information such as antenna radiation patterns, the rendering of wired links, and traffic flow information. Other non-networking visual enhancements are also planned, such as support for text rendering to display node names, adding a 3D compass to help the user navigate through a scenario, or being able to select a node/link directly from the scene. This also includes expanding the catalog of available 3D object models to include more generic models of simple shapes and some network-specific models such as cell towers and Wi-Fi router models so users do not have to worry about creating or sharing their own models.

Finally, we also have plans to add an online mode, allowing visualization while the scenario is currently running, similar to the functionality provided by PyViz, but with the ease of integration provided by NetSimulzyzer.

## 5 CONCLUSION

In this paper we presented a new open-source visualizer for ns-3 to fill the gaps of other existing visualization and animation tools. The 3D scene renderer provides the user with the ability to navigate through the topology and follow nodes' movements. The flexible plotting architecture allows users to select and configure the plots they want to show, and extend the capabilities to other types of plots. Using several examples, we demonstrated how this visualizer can help researchers understand their scenarios and share their results with the community. Finally, we discussed on-going work to further extend the supported features, which we hope can be done in collaboration with the ns-3 user community.

## ACKNOWLEDGEMENTS

The authors would like to thank Tom Henderson for providing technical insights on ns-3 architecture and feedback throughout the development of this tool.

## REFERENCES

- [1] Hany Assasa, Joerg Widmer, Tanguy Ropitault, and Nada Golmie. 2019. Enhancing the ns-3 IEEE 802.11ad Model Fidelity: Beam Codebooks, Multi-Antenna Beamforming Training, and Quasi-Deterministic MmWave Channel. In *Proceedings of the 2019 Workshop on ns-3* (Florence, Italy) (WNS3 2019). Association for Computing Machinery, New York, NY, USA, 33–40. <https://doi.org/10.1145/3321349.3321354>

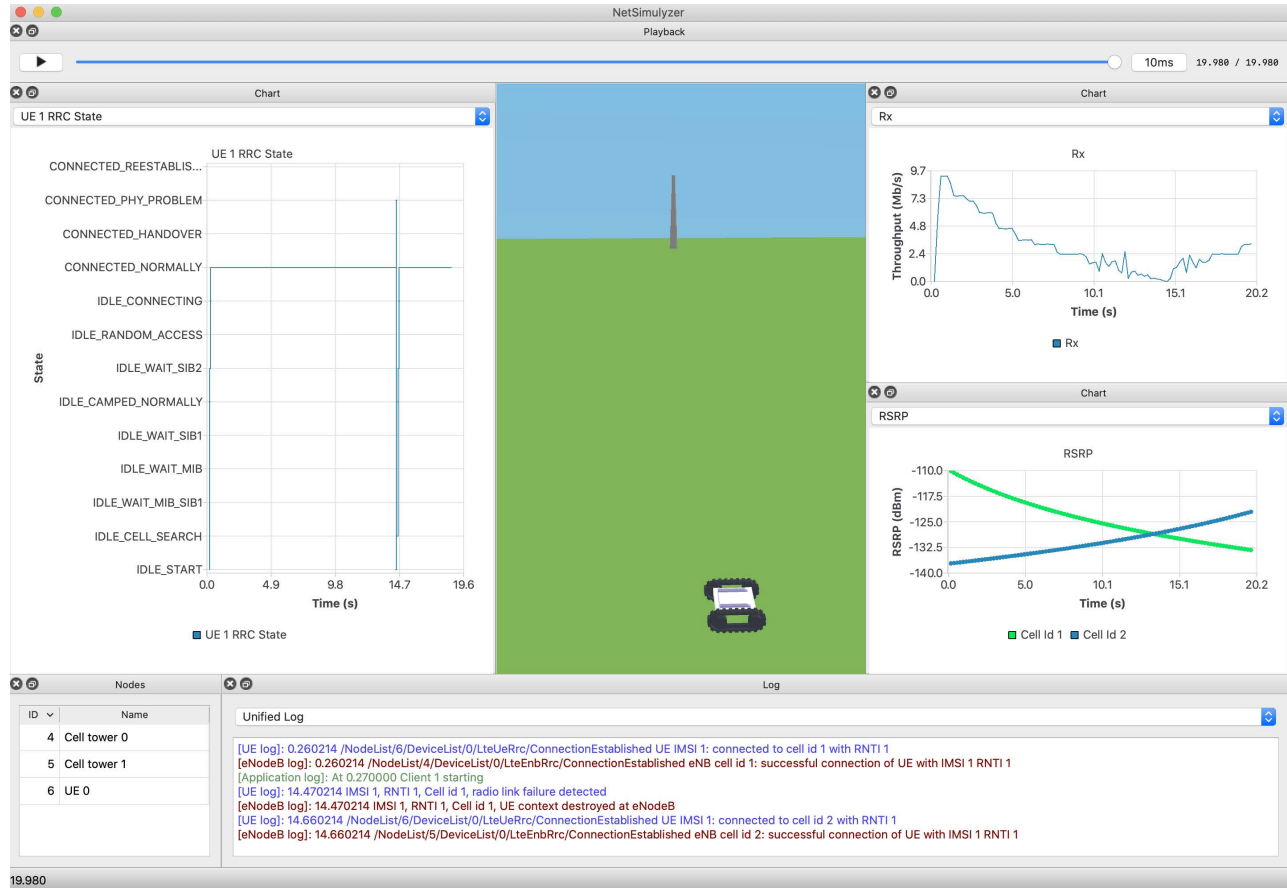


Figure 5: Visualization of lna-radio-link-failure Scenario

- [2] Oscar G. Bautista and Kemal Akkaya. 2020. Extending IEEE 802.11s Mesh Routing for 3-D Mobile Drone Applications in ns-3. In *Proceedings of the 2020 Workshop on ns-3* (Gaithersburg, MD, USA) (WNS3 2020). Association for Computing Machinery, New York, NY, USA, 25–32. <https://doi.org/10.1145/3389400.3389406>
- [3] Giuseppe Bianchi. 2000. Performance Analysis of the IEEE 802.11 Distributed Coordination Function. *IEEE Journal on Selected Areas in Communications* 18, 3 (2000), 535–547. <https://doi.org/10.1109/49.840210>
- [4] NetAnim. 2017. <https://www.nsnam.org/wiki/NetAnim>.
- [5] Ben Newton, Jay Aikat, and Kevin Jeffay. 2015. Simulating Large-Scale Airborne Networks with ns-3. In *Proceedings of the 2015 Workshop on ns-3* (Barcelona, Spain) (WNS3 2015). Association for Computing Machinery, New York, NY, USA, 32–39. <https://doi.org/10.1145/2756509.2756514>
- [6] NIST. 2017. Clear Talk for First Responders. NIST Modeling Tool to Help Advance Cellular Emergency Communications. <https://www.nist.gov/news-events/news/2017/10/clear-talk-first-responders>.
- [7] NIST. 2021. NetSimulyzer ns-3 Module, v1.0.1. <https://github.com/usnistgov/NetSimulyzer-ns3-module>.
- [8] NIST. 2021. NetSimulyzer Visualizer Application, v1.0.1. <https://github.com/usnistgov/NetSimulyzer>.
- [9] ns-3 Network Simulator. 2021. <https://www.nsnam.org/>.
- [10] L. Felipe Perrone, Thomas R. Henderson, Mitchell J. Watrous, and Vinicius Daly Felizardo. 2013. The Design of an Output Data Collection Framework for ns-3. In *2013 Winter Simulations Conference (WSC)* (Washington, DC, USA). 2984–2995. <https://doi.org/10.1109/WSC.2013.6721666>
- [11] PyViz. 2015. <https://www.nsnam.org/wiki/PyViz>.
- [12] Paulo Alexandre Regis, Suman Bhunia, and Shamik Sengupta. 2016. Implementation of 3D Obstacle Compliant Mobility Models for UAV Networks in ns-3. In *Proceedings of the Workshop on ns-3* (Seattle, WA, USA) (WNS3 2016). Association for Computing Machinery, New York, NY, USA, 124–131. <https://doi.org/10.1145/2915371.2915384>
- [13] RemCom. 2021. EMPIRE Shim. <https://apps.nsnam.org/app/empire/>.
- [14] Richard Rouil, Evan Black, Samantha Gamboa, Wesley Garey, and Thomas Henderson. 2020. Simulation and Visualization of Public Safety Incidents. <https://www.nist.gov/ctl/pscr/simulation-and-visualization-public-safety-incidents>.
- [15] Benjamin Sliwa, Manuel Patchou, Karsten Heimann, and Christian Wietfeld. 2020. Simulating Hybrid Aerial- and Ground-Based Vehicular Networks with ns-3 and LIMOsim. In *Proceedings of the 2020 Workshop on ns-3* (Gaithersburg, MD, USA) (WNS3 2020). Association for Computing Machinery, New York, NY, USA, 1–8. <https://doi.org/10.1145/3389400.3389407>
- [16] US Naval Research Laboratory. 2021. Scripted Display Tool (SDT), a 3D Network Visualization Tool, v2.3. <https://github.com/USNavalResearchLaboratory/sdt>.
- [17] Amina Šljivo, Dwight Kerkhove, Ingrid Moerman, Eli De Poorter, and Jeroen Hoebeke. 2018. Interactive Web Visualizer for IEEE 802.11ah ns-3 Module. In *Proceedings of the 10th Workshop on ns-3* (Surathkal, India) (WNS3 2018). Association for Computing Machinery, New York, NY, USA, 23–29. <https://doi.org/10.1145/3199902.3199904>
- [18] Tommaso Zugno, Michele Polese, Natale Patriciello, Biljana Bojović, Sandra Lagen, and Michele Zorzi. 2020. Implementation of a Spatial Channel Model for ns-3. In *Proceedings of the 2020 Workshop on ns-3* (Gaithersburg, MD, USA) (WNS3 2020). Association for Computing Machinery, New York, NY, USA, 49–56. <https://doi.org/10.1145/3389400.3389401>