Mininet Topology Generator

Project Report

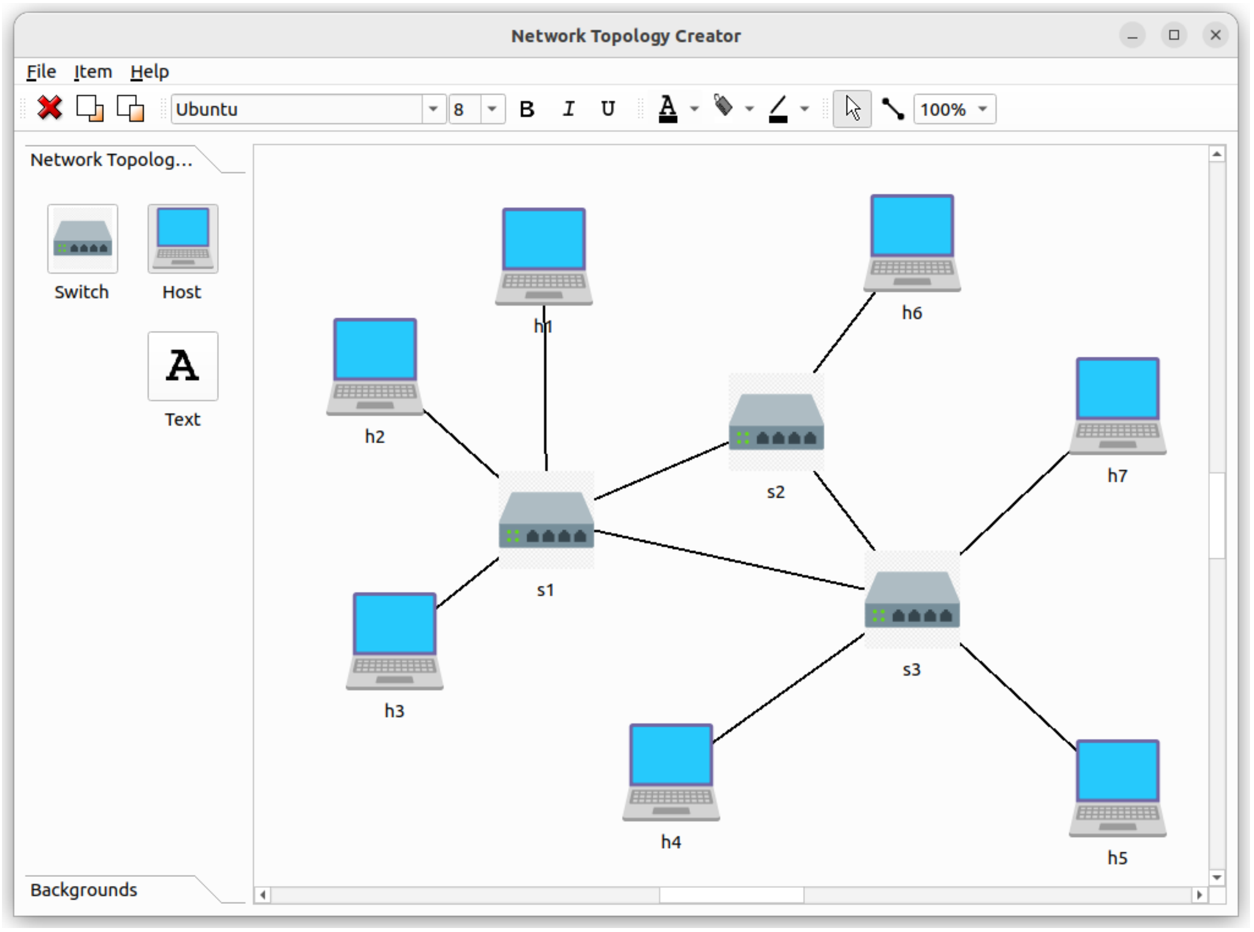
Ivan Mendoza

Pablo Sotelo Torres

Luis Garza Garcia

Project Summary

Our team was tasked with creating a Mininet Topology Generator that allows users to design network topologies through a graphical interface and simulate them using Mininet. The final product consists of a GUI built in Python/Qt that allows users to manually draw or generate topologies, which are saved as JSON files and passed to Mininet for simulation. Alternatively, the user can load a topology from a JSON file and visualize it on the canvas.



- Ivan Mendoza: Was in charge of the file transfer between Python/Qt, and Mininet via JSON, as well as generating the function to visualize the layout of the topology with Kamada-Kawai using *NetworkX*, and adapted it to extract coordinate data necessary for GUI visualization. He also handled quality of life features, like auto-zoom and topology generation from text input (subnets, flat structures), and handled various debugging tasks.

- Pablo Sotelo Torres: Created and adapted the GUI using a premade Python/Qt template from Qt example page. After that, he integrated save and load to and from JOSN, respectively. He integrated visualization features using Ivan's layout code, ensuring the interface could properly load and display topologies from saved JSON files. Finally, he integrated features that Luis created; that is Mininet Run function.

- Luis Garza Garcia: Handled the Mininet backend, writing the code that reads JSON topologies and instantiates them as functional virtual networks in Mininet through a python script. He also implemented a function to launch an *xterm* terminal window, enabling real-time interaction with the simulated networks.

Project Timeline/Plan

- **Week 1:**
    - Ivan developed a demo, it simulated a GUI using text prompts, the code asked the user for the amount of hosts, and switches, however it connected all hosts to switch 1, and saved the topology to a JSON format decided on by Ivan.
    - Pablo started searching for possible templates to build upon and found diagramscene.py, which he used as the foundation of the GUI.
    - Luis developed a python script to create custom network in python and launch and test the custom network.

- **Week 2:**
    - Ivan adapted the JSON format suggested by Dr. McGarry, he implemented subnet text generation, as well as Manual generation, where the user can specify specific connections in order to better simulate GUI interaction. Ivan also developed a function that extracted each unique instance of a node.
    - Pablo initiated the integration of the delete operation, added icons for hosts and switches, and enabled drawing and editing functionalities.

    - Luis modified the python script so it would read the links from a JSON file and create and run ping all command to test the custom network connections, however it only worked with simple topologies.

- **Week 3:**
    - Ivan created a function using Kamada-Kawai, that took coordinates in a topology, and from the coordinates, created a visual representation of the topology.
    - Pablo fixed display errors on canvas with host and switch images, removed the arrows between icons, and integrated the save function.
    - Luis fixed the python script so that it works with any custom topology not only simple topologies but with any topology as long as the links are in a JSON file.

- **Week 4:**
    - Ivan implemented previous demo features, text-based generation, which allows a user to create subnet and flat topologies from text.
    - Pablo implemented the Load function using Kamada-Kawai, which Ivan later polished as described above. Pablo also integrated Luis's function to run Mininet alongside the GUI.
    - Luis modified the python script to open the Mininet command line interface (CLI) with the custom topology in a new xterm terminal

Test Plan/Test Results

- Visual Layering Bug: Links were being rendered above host/switch icons when loading a topology. Ivan fixed this by adjusting the rendering order.

- Zoom Scaling Bug: Loaded topologies sometimes appeared too large to fir within the canvas, which required manual zooming. Ivan implemented an autozoom feature to dynamically adjust the view to fit the entire topology.

- Node Placement usability: Initially, users had to reselect icons from the palette every time they wanted to place a new host/switch. Ivan modified the behavior to allow multiple placements after a single selection.

- Clear Canvas: Ivan added a clear canvas feature to allow users to remove all elements from the canvas without restarting the application or manually deleting the icons.

- Test the connections of the custom topology running *pingall* in the *xterm* terminal using the Mininet CLI. Verifying the connections between all hosts in the network.

Project Future

This project could be expanded in a number of ways:

- Add the option for the user to change the parameters of the links (Bandwidth, delay, max queue length and packet loss)

- Add a button to run *iperf* in the selected host by the user to test the performance of the network.

- Add predefined network architectures options for easy creation (simple, linear, tree, ring, start, mesh, etc.)

- Implement the ability to erase and readjust icon naming and numbering for already-placed elements. Currently, if icons like h1 and h2 are deleted, the next added host will be named h3 instead of reusing available identifiers. Future work will address this by ensuring consistent and correct numbering, preventing unnecessary gaps in naming.

- A scissor tool that can delete icons by clicking on them. Being able to create links by clicking on two icons without needing to drag the link to it. Undo/Redo functionality. Resizable canvas. Snap to grid options

- Show the output of the *pingall* without the need for the *xterm* window to pop up, a simple interaction for when simplicity is important

- Packet simulation animation, to visualize packet travel between links, can be affected by speed, loss, and delay of travel