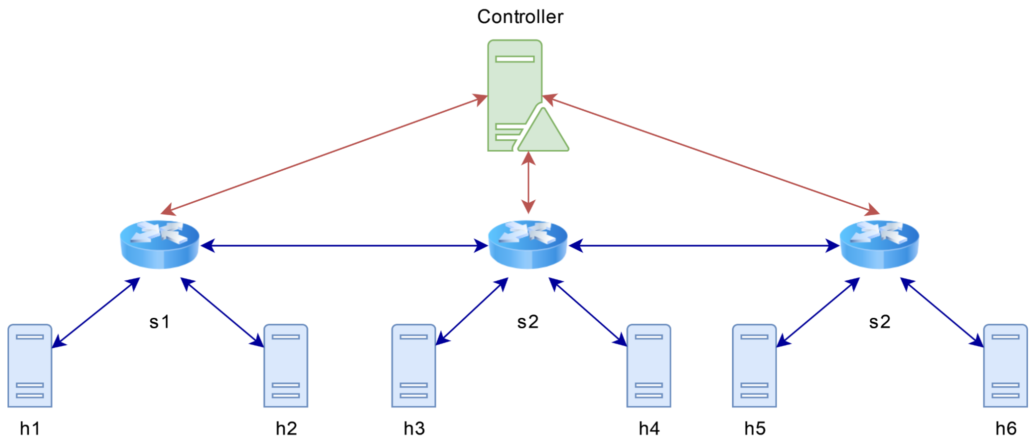
1. The network topology/topologies you plan to simulate (number of switches, hosts and links):

The simulation environment can be customized to accommodate varying numbers of hosts and links, based on the requirements. We will be simulating a network topology consisting of 3 switches and multiple hosts (h1 to h5) connected through ethernet links. We add 11 ethernet links and a controller connected to the switches. The virtual network can handle as much data as the computer it's running on can handle according to mininet. So, to make the experiment easier we can limit the data bandwidth capacity for each link in the topology. To support our network topology, we have created a diagram using draw.io, which can also be created through mininet cmd or python code for the technical aspect of the simulation.



2. The controller you would be using:

We have decided to utilize the Ryu controller, an open-source software-defined networking (SDN) controller. We have chosen Ryu due to its user-friendly interface and extensive range of features. Another key advantage of using Ryu is its compatibility with Mininet, a network topology simulation tool. This compatibility allows us to develop and test sophisticated network applications using the Python programming language within a simulated environment using both Mininet and Ryu (https://ryu.readthedocs.io/en/latest/api_ref.html).

3. High-Level Modular Design:

Our software will be designed to monitor network activity and identify any unusual patterns that could indicate a DDoS attack. The modular structure will consist of the following scripts:

- The first script will be responsible for implementing our network topology.
- In the second script, all normal events will be monitored and tracked (Packet Catch Module).
- The third script will be used to launch the attacks (Attack Module), we also need to launch normal requests as part of ML training, note that these steps could be done via cmd.
- The fourth script will gather the necessary data to train the model (Training Module).
- The fifth script will use the data to feed the ML model for testing purposes (Module for inspecting packets). If an attack is found, this will trigger an alert (Alert Module).

4. External Applications:

- Wireshark: Real-time capture and analysis of network traffic will be done with Wireshark.
- Pycharm/Spyder: Could be used to run python scripts rather than cmd.

Libraries/APIs:

- Telegraf: For collecting the data to feed into the ML model (https://docs.influxdata.com/telegraf/v1.13/).
- Influxdb: For storing the data generated (https://docs.influxdata.com/influxdb/v1.7/).
- Hping3: For flooding the network with ICMP packets (https://www.kali.org/tools/hping3/)

5. Test Plan:

1. Design the network topology according to the specifications.
2. Launch the controller to manage the network devices.
3. Test the network connectivity by performing a ping test on the hosts using ICMP.
4. Generate network traffic by flooding the network with hping3 (Attack Module) while also conducting normal ping tests. Both types of tests are necessary for training the ML model.

5. Capture the required data using telegraf (Packet Catch Module). The data should include changes within the network and the number of ICMP messages with their corresponding flags (0 for normal and 1 for attack).
6. Store the collected data in the Influxdb database for easy retrieval.
7. Convert the data retrieved from Influxdb into a readable format, and use a pre-trained model to detect and flag any attacks (Module for inspecting packets).
8. Trigger an alert when an attack is detected by the machine learning model (Alert Module).
9. Block requests from the source IP address running the pings to prevent further attacks.
10. Compare the accuracy of various machine learning models to select the best model for the task.

    Note that the AI algorithm will continually analyze new packets and repeat steps 5-9.