The Maple system, developed and explained in a recent paper by Voellmy, Wang, Yang, Ford and Hudak, provides a simplified, abstracted and scalable approach to software-defined networking (SDN). While the well-known OpenFlow protocol provides a universal method for routing packets from a programmable switch (which is managed by a separate controller), Maple allows the user to define a simpler routing algorithm than the complex rule-based systems that can arise in large, dynamic networks. This is done through the observation of a single function (titled $f$ in the relevant paper); the construction of a trace tree to represent routing rules, the optimization of the trace tree to simplify these rules; and the conversion of the tree to OpenFlow rules that can be sent to a given switch.

IntelliJ IDEA, meanwhile, is a popular integrated development environment (IDE) for Java that is available for both personal and commercial development. While IntelliJ is not open source, it does support the use of plugins, which can add to the functionality of IntelliJ. There are currently 2099 plugins available for IntelliJ, and programmers can also develop their own to add more specific functions to the programming environment.

In my senior project, I hope to develop a plugin for IntelliJ (or Eclipse, if that turns out to be better suited for this project) that allows users to program their own Maple network with as many of the details as possible taken care of by the plugin. Currently an implementation of such

---


a plugin exists for Cloud9, an IDE that runs in a virtual machine. However, this development environment is very limited, and it is difficult for users to test and debug with the current implementation.

Developing a plugin requires four steps. First, we must set up the programming environment for users inside IntelliJ, which includes following online instructions for setting up an IntelliJ plugin and creating a simple step-by-step process that allows users with the plugin to set up a Maple environment. Second, we must write code that implements every part of the Maple system with the expect of the single function $f$, which can be defined by the user. Much of this is already done in the Cloud9 plugin, so I will need to learn the way the Cloud9 plugin works inside and out before transferring these features over to IntelliJ. Third, we need to create a method to deploy the system on a network, so that users can actually use the plugin to program switches on their machine. The current Cloud9 plugin does this with a mininet network and OpenDaylight controller, but I will consult with Professor Yang to determine the best deployment method for the IntelliJ plugin’s use case. Fourth, the plugin will need some visualization of the network state. This information is important to the user both for analysis during the (successful) running of a controller and for debugging, when the network is not functioning correctly. This is one area where the Cloud9 plugin is lacking, so hopefully the plugin I develop will make improvements in this area.

One challenge I will face in this last step is that debugging a Maple network with this plugin is different from debugging code in general. Most of the time the function $f$ is run only conceptually, through the OpenFlow flow tables, and not literally. If I can get to this stage of the project, I would like to develop a debugging tool that meshes well with the way Maple works.

---

For example, one idea is to develop a tool that efficiently watches all decisions made at the switches, so that we can stop the program when a packet is sent the wrong way and see what the flow tables are at that point, as well as when they were recently changed and what made them change.

While developing a plugin of this size and complexity may take a long time, I will get as far as I can under the guidance of Professor Yang. The scope and steps of the project may change as challenges are identified and tackled, but I plan to submit the following deliverables at the end of the semester:

- All source code, including a plugin that works on the IDE we have selected
- A final report with the following information:
  - Motivation behind this project
  - Features of the plugin, and the role I played in their origination and development
  - Steps required to run the plugin
  - Challenges I faced while developing this plugin, both solved and unsolved
  - Future work that can be done by someone who wants to continue development of the plugin