CS490 Project Proposal
Lewn Yu
Advisor: Yang R. Yang

Abstract
Software-defined networking (SDN) offers a centralized, programmable approach to define network behaviors. OpenFlow (an SDN-enabling protocol that is gaining industry traction) require programmers to manage flow tables across its switches, which implies a distributed approach that is at odds with the ideal of a centralized controller. Maple is a powerful OpenFlow controller that allows a single algorithmic policy to define the network flow, without needing to manage flow tables and program individual switches. The current Maple prototype has already achieved impressive simulated and real benchmarking results, and does so with a powerful optimizer and scheduler. For my research project, I will assist with the further improvement and development of Maple.

Background
According to Wikipedia, The principle of SDN is based on decoupling the network’s control plane (the system that decides where traffic is sent) from the network’s data plane...
(the system that forwards traffic). This architecture enables programmers and network administrators to control and shape network traffic from a centralized control plane, without needing to tamper with actual network hardware.

The OpenFlow Specification is an open standard hosted at Stanford University. OpenFlow defines flow tables for each switch (each flow entry contains packet fields for matching and a resulting action for matching packets) as the data-plane abstraction. For matched packets, the corresponding action in the flow table is performed. For unmatched packets, the switch sends it to a central controller, which can then decide to forward, drop or add a new flow entry for the packet. The controller can also query the state of switches. The OpenFlow standard has played a critical role in the development and study of SDNs. At the time of this report, there are a number of OpenFlow controller implementations, such as NOX, Maestro, Beacon, Trema, and Nettle, among many others.

Maple, an OpenFlow controller based on a single programmable algorithmic policy, is implemented in such a way that performance and scalability impacts are minimal. In particular, it optimizes decision locality by constructing trace trees to effectively cache results of the algorithm for particular input packets. These trace trees can then be used to construct distributed OpenFlow tables at switches to improve performance. The distributed flow tables are updated every time the network topology changes.

---

optimizations are carried out transparently to the programmer, who only needs to design and write the core controller policy.

In addition to the optimization component, there is a scheduling component that implements switch-level parallelism and batched message processing. The scheduler scales the controller across multiple cores to improve performance.

The end result is a scalable and performant OpenFlow controller that allows the SDN programmer to define a single algorithmic policy \( f \), which will then in principle be applied to all incoming packets (though, as described, in reality this is minimized with the use of trace trees and distributed flow tables). This maintains the essence of SDN by exposing what is effectively a truly centralized controller to the programmer.

*Figure 3: simplified architecture of Maple. The SDN programmer only needs to define the algorithmic policy \( f \). The flow tables are generated by the optimizer.*
Deliverables

1) Code contributions
2) Log of bug fixes (if applicable)
3) Experiment statistics
4) Report

Discussion of Plans

In the current prototype of Maple, there are three key features: trace tree construction, transparent policy distribution, and scalable multicore scheduling. The prototype is implemented in Haskell. Therefore, I will spend some time early in the project to intensely study relevant portions of the Glasgow Haskell Compiler, as well as relevant portions of the Linux kernel that Maple interacts with. Tentatively, I will work on portions of the multicore scheduler, but I may end up working on the optimizer as well. Since this is an expansive systems project, my work will likely deviate from what I described in this proposal, but will be defined by my advisor Yang R. Yang and included in the final report. Below is my tentative plan:

1) Ramp up on background (SDNs, OpenFlow interface, Nettle)
2) Closely study Maple, Haskell, GHC, Haskell run-time system
3) Study the Maple codebase
4) Write code, fix bugs, run tests
5) Report findings