Stateful Software Defined Networking and Its Applications: a CS490 Project Proposal

Henry Davidge   Adviser: Richard Yang

February 5, 2014

Motivation

In the past several years, Software-Defined Networking (henceforth referred to as SDN) has emerged as one of the most exciting topics in the field on computer networking. Rather than write complex, low-level distributed flow rules, SDN allows users to design familiarly algorithmic network policies with a global view of network state. The OpenFlow standard provides a widely accepted protocol for the centralized controller to push policies to individual network switches in the form of flow tables, which map properties such source IP address to packet actions. Such technology allows for a separation of the control plane, where the users define their high-level algorithmic policies, and the data plane, where switches interact with individual packets.

This level of abstraction becomes particularly useful in dynamic environments such as the modern heterogeneous data center. Cloud services such Amazon Web Services provide users with pools of virtual rather than physical servers that provide a vast area of services. This diversity of network function in addition to the need for fault tolerance and efficiency can lead to network topologies that are both highly complex and highly rapidly changing, providing motivation for the level of abstraction and programmability provided by SDN.

However, while current SDN programming techniques provide much more power and simplicity than traditional network switch programming, a realistic SDN framework must provide an expressive way to handle changes in network state. To return to the data center example, by providing virtual machines rather than physical servers cloud computing providers are both able and expected to provide extremely high reliability. If the physical machine hosting a VM happens to fail, the provider needs to quickly allocate another machine and proceed to route all future traffic to the new address. In the context of SDN, this redirection requires the invalidation of the flow rules leading to the old host, a calculation of the new shortest path, and the installation of corresponding policies at the switch level.

The efficiency with which these operations are carried out can have drastic implications on the performance of the network. If a switch’s local flow tables do not contain enough information to determine the correct forwarding destination of an incoming packet, the packet
must be sent to the controller and thereby incur significant communication costs. On the other hand, expecting the programmer to provide extensive contingency plans for global state changes can lead to undo burden on the developer. The SDN framework of the future must reconcile these two competing forces and provide a user-friendly system that can efficiently and intelligently handle changes in global network state.

Applications

In the previous section we mentioned efficient fault tolerance as an area where SDN could provide significant benefits over traditional networking techniques. However, given the presence of state-cognizant, more and more functionality can be transferred to the network layer. If the controller can effectively adapt to changes in the network state, this delegation of responsibility can lead to significant payoffs in performance, cost, and administrative burden.

Virtualized Network Functions (VNF)

One of the great promises of Software Defined Networking is the opportunity to take traditional networking functions that rely on expensive proprietary hardware and execute them in software using generic programmable network switches. These applications are known as Virtualized Network Functions (VNFs), and are responsible for much of the excitement around SDN. Such a shift would allow network administrators significant flexibility to iterate quickly on system design and install new components with much less pain.

For example, let us investigate Network Address Translation (NAT). A NAT service allows a number of machines, perhaps even a very large number in the context of a data center, to share a single external IP address. The NAT uses local IP and port information to multiplex many concurrent connections using the single external address. Despite the relative simplicity of this task, in large data centers NAT functionality can be quite expensive, requiring either dedicated hardware or a costly router. However, if a NAT service were efficiently designed and implemented as a VNF, it could run alongside other network functions on generic OpenFlow compatible switches. Although the ease with which one can conceptualize a NAT does not necessarily lead to inexpensive hardware, we hope that the simplicity will enable us to efficiently prototype such a service using SDN.

A robust load balancing service stretches the capabilities of virtual networking a bit farther. Such a service must be aware of much information about the global network state – at any moment in time, it must be aware of which machines are operational, the applications running on each machine, and the load on each node. This information must be gathered in real time and stored in such a way that the system can quickly detect changes in state and devise new corresponding flow capabilities. However, the reward of such a system reflects the complexity. If the load balancing policies can be efficiently calculated at the controller and delivered to the switches, the load balancer can be effectively baked into the network, providing enhanced performance and greater sensitivity to the rapid change of application traffic inherent in modern heterogeneous data centers.
Foundations and Goals

Recent research in Prof. Yang’s group has provided many of the fundamentals necessary for these higher level applications. *Maple* provides an abstraction that allows the programmer to design algorithmic policies that are automatically translated into flow tables, while *Mercury* begins to establish an in-memory *Network Database* that allows the controller to store information about global network state and the programmer to add callbacks to changes. My goal for this semester are to design and implement a set of SDN applications that demonstrate the robustness and utility of this new technology. While we have not completely decided on which applications I will focus on, one will likely be a load balancer as I find the implications of such a utility very interesting in the context of large data centers. In addition, I may very well have to work on improving the ability of existing systems to deal with querying and storing network information in order to build these higher level applications. At present we plan to have most of the design and implementation completed and ready for demo by the beginning of March.

Deliverables

At the end of the semester I plan on delivering a report detailing the design of my services as well as their performance on realistic workloads. The report will focus on both the traditional algorithmic difficulties of such applications as well as the unique systems challenges we will inevitably face with young and non-traditional programming frameworks.