MIT 6.829: Computer Networks Fall 2017

Lecture 17: Infrastructure as a Service: SDN's, w/ Daniel Firestone

Scribe: Jonathan Kelly, Quanquan Liu, Miranda McClellan

1 Infrastructure as a Service

How does Microsoft support approximately 1 million virtual networks, spread over a region of about 1 million server data centers?

1.1 Software Defined Networking: Building the Right Abstractions

There are three layers in the abstraction of the software defined networking:

1. Management
2. Control
3. Data

One approach to designing an infrastructure that supports the three different levels of abstraction is to separate out the different parts into different planes (i.e. the management plane, the control plane, and the data plane). An example separation of the three layers of abstraction into access control lists (ACLs) is the following:

1. Management plane: Create a tenant.
2. Control plane: Pub tenant ACLS to switches.
3. Apply ACLs to flows.

One requirement for the data plane is that it needs to apply billions of flow policy actions to packets.

For example, a different policy might be needed for each switch.

Meanwhile the actual equipment can't hold all the policy actions in a single chip. Thus, we need to store policy in virtual switches.

1.2 Example 1: Load Balancing

All the infrastructure runs behind a load balancer to enable high availability and application scale.

How do we make application load balancing scale to the cloud?

How do you load balance the load balancers?

Hardware load balancers are expensive and cannot support the rapid changes in policies.

What should a load balancer do?

The goal of a load balancer is to map Virtual IP addresses to a Dynamic IP address set in a cloud service. This needs to be done on a per-TCP connection. There are two steps to this process:

1. Load balance by selecting a DIP.
2. Use a NAT function to map a VIP to a DIP and ports.

We push the NAT function to all the MUX switches while a single controller abstracts out the process of assigning load balancers.

The management plane then takes the physical description of the network, the controller defines all the data path elements (i.e. the Azure VMSwitches), and, finally, the switches assign the final policies (i.e. the VL2 policies).

In essence, what is accomplished is that the data policies are abstracted out from the control switches. The controller finds relevant switches and find all the relevant data planes to program the current data policy.

Virtual switches are everywhere. This scheme abstracts out how network is managed from how data is actually transferred in the network.

1.3 Microsoft Azure Virtual Switch Design

Since big companies such as Amazon, Microsoft and Google didn't want to constantly wait on the discretion of Cisco to get new networking upgrades, they decided to build their own customizable vswitches.

Early approaches to virtual switch (2009-2011): Each SDN application is a driver module hard compiled into the vswitch, handling packets on its own. Any SDN application updates require kernel updates also.

Goals for Azure Host SDN Platform:

1. Goal 1: Provide a programming model allowing for multiple simultaneous, independent network controllers to program network applications, minimizing cross-controller dependencies.
2. Goal 2: Provide a MAT programming model capable of using connections as a base primitive, rather than just packets – stateful rules as first class objects. Need to keep a state of what programming actions are in the connection.
3. Goal 3: Provide a programming model that allows controllers to define their own policy and actions, rather than implementing fixed sets of network policies for predefined scenarios.
4. Goal 4: Provide a serviceability model allowing for frequent deployments and updates without requiring reboots or interrupting VM connectivity.

2 Virtual Filtering Platform (VFP) Azure's SDN Dataplane

The basic module is to allow controllers to layer policy on top of each other. One takes input from the VM and translate to policy in the VFP layer. All policy is in the controller and the VFP is a fast, flexible implementation of the policy.

Three primitives exposed to controllers:

1. Layers
2. Rule Matches
3. Rule Actions

Controllers program all of the switches and all of the tables in the switches. Vswitches don't know that they're constructing LBs or a distributed application, only the controller knows, switches just match rules to actions.

MAT (Match Action Tables) VFP exposes a typed MAT API to the agents/controllers. One table ('layer') per policy. Each table is entirely programmable by the controller. Each table reroutes the data packets to the correct IP addresses and match rules from the controllers to a data action. Layers do not know any of the data policy rules in the controller but instead find out these rules from the controller via the MAT. Applications don't know NAT rules or access control lists. Flow table maintains all previous data flow interactions.

There exists lots of latency in delivering the policies; therefore, we need some way of managing the latency. Some concept of most recently used policies.

Since there are two-way TCP flows, everything is stateful. This makes it easy to reason about asmmetric policy.

3 Internal Load Balancing Example

Assume that the IP address is from an external source. Create entire new load balancing layer without changing anything in the kernel. Controller can create new functions on the fly without making hardware changes to create new network functions.

Need new primitives for actions. ASIC pipeline model: parse once, modify once. Parse flowID and headers and match to each of the layers, and then match on the other side.

Header actions: add to the header, edit the header, and delete something in the header. Header transposition: table of things to modify. Combine actions as a single action. Header by header and transposition by batches.

Policy bottlenecks (vs. when not applying this switching policy): Where do such bottlenecks occur? Original software introduced much more congrestion and delays. Rule lookups and hash lookups are decoupled to improve performance. Cache all the first packet lookups in the rules table, for every subsequent packet only need to perform a hash table lookup which is a lot less expensive.

Current bottleneck: processing packets in the switches, need to process the packets in the network stack and look up the rules per packet. Hardware constraints prevent the packets from being handled more effectively.

How do we scale the controllers?

Controller scaling hierarchically. Have a regional network manager, a cluster network manager, and a host that manages the cluster managers.

How do you communicate with the controller?

The controllers are just VMs in the network with their own IP addresses. Basic VMs can be communicated with directly.

Azure uses Paxos –standard distributed system architecture.

Most interesting point: Doing all the networking on the host makes the physical network fast and scalable. Everything is the software (on the host and in the VM). Key point of software defined networking: the physical network becomes very scalable. 50x improvement in the network throughput but not a 50x improvement in the CPU power.

Traditional approaching to scaling: use ASICs. Is this a long term solution? Example ASIC solution: no use of SDN policy, network is limited by the hardware.

Solution to the hardware bottleneck: Azure SmartNIC (FPGA): use programmable chips. FPGA is a giant array of programmable gates, and lay out the programmable chips accordingly on the chip.