Programming SDNs

- The Good
  - Network-wide visibility
  - Direct control over the switches
  - Simple data-plane abstraction

- The Bad
  - Low-level programming interface
  - Functionality tied to hardware
  - Explicit resource control

- The Ugly
  - Non-modular, non-compositional
  - Challenging distributed programming

Network Control Loop

Read state

Compute Policy

Write policy

OpenFlow Switches

Language-Based Abstractions

Query abstractions

Writing/combining modules

Update abstractions

OpenFlow Switches

Policy in OpenFlow

- Defining “policy” is complicated
  - All rules in all switches
  - Packet-in handlers
  - Polling of counters

- Programming “policy” is error-prone
  - Duplication between rules and handlers
  - Frequent changes in policy (e.g., flowmods)
  - Policy changes affect packets in flight

Policy as a Function
From Rules to a Policy Function

- Located packet
  - A packet and its location (switch and port)
- Policy function
  - From located packet to set of located packets
- Examples
  - Original packet: identity
  - Drop the packet: none
  - Modified header: modify(f=v)
  - New location: fwd(a)

From Bit Patterns to Predicates

- OpenFlow
  - No direct way to specify dstip!10.0.0.1
  - Requires two prioritized bitmatches
    - Higher priority: dstip=10.0.0.1
    - Lower priority: *
- Using boolean predicates
  - Providing & | and ~
  - E.g., ~match(dstip=10.0.0.1)

Virtual Header Fields

- Unified abstraction
  - Real headers: dstip, srcport, ...
  - Packet location: switch and port
  - User-defined: e.g., traffic_class
- Simple operations
  - Match: match(f=v)
  - Modify: modify(f=v)
- Example
  - match(switch=A) & match(dstip='1.0.0.3')

Queries as Buckets

- Forwarding to a “bucket”
  - Q = packets(limit=1,group_by=['srcip'])
- Callback functions
  - Q.register_callback(printer)
- Multiple kinds of buckets
  - Packets: with limit on number
  - Packet counts: with time interval
  - Byte counts: with time interval

Power of Policy as a Function

- Dynamic policy
  - A stream of policy functions
- Composition
  - Parallel: Monitor + Route
  - Sequential: Firewall >> Route

Computing Policy

Parallel and Sequential Composition
Topology Abstraction
Combining Many Networking Tasks

Monolithic application

Controller Platform

Hard to program, test, debug, reuse, port, ...

Modular Controller Applications

A module for each task

Controller Platform

Easier to program, test, and debug
Greater reusability and portability

Beyond Multi-Tenancy

Each module controls a different portion of the traffic

Controller Platform

Relatively easy to partition rule space, link bandwidth, and network events across modules

Modules Affect the Same Traffic

Each module partially specifies the handling of the traffic

Controller Platform

How to combine modules into a complete application?

Parallel Composition

srcip = 5.6.7.8 \rightarrow count
dstip = 1.2.3.4 \rightarrow fwd(1)
dstip = 3.4.5.6 \rightarrow fwd(2)

Controller Platform

Parallel Composition

srcip = 5.6.7.8 \rightarrow count
dstip = 1.2.3.4 \rightarrow fwd(1)
dstip = 3.4.5.6 \rightarrow fwd(2)

Controller Platform

srcip = 5.6.7.8, dstip = 1.2.3.4 \rightarrow fwd(1), count
srcip = 5.6.7.8, dstip = 3.4.5.6 \rightarrow fwd(2), count
srcip = 5.6.7.8 \rightarrow count
dstip = 1.2.3.4 \rightarrow fwd(1)
dstip = 3.4.5.6 \rightarrow fwd(2)
Sequential Composition

Controller Platform
Routing
Load Balancer

\[
\begin{align*}
\text{srcip} &= 0^*, \text{dstip} = 1.2.3.4 \rightarrow \text{dstip} = 10.0.0.1, \\
\text{srcip} &= 1^*, \text{dstip} = 1.2.3.4 \rightarrow \text{dstip} = 10.0.0.2, \\
\text{dstip} &= 10.0.0.1 \rightarrow \text{fwd}(1), \\
\text{dstip} &= 10.0.0.2 \rightarrow \text{fwd}(2).
\end{align*}
\]

Dividing the Traffic Over Modules

- **Predicates**
  - Specify which traffic traverses which modules
  - Based on input port and packet-header fields

  - **Web traffic**
    - \( \text{dstport} = 80 \)

  - **Non-web traffic**
    - \( \text{dstport} \neq 80 \)

Abstract Topology: Load Balancer

- **Present an abstract topology**
  - Information hiding: limit what a module sees
  - Protection: limit what a module does
  - Abstraction: present a familiar interface

Abstract Topology: Gateway

- **Left**: Learning switch on MAC addresses
- **Middle**: ARP on gateway, plus simple repeater
- **Right**: Shortest-path forwarding on IP prefixes
High-Level Architecture

Controller Platform

M1 M2 M3 Main Program

Questions

• Other ways to combine multiple policies?
• How to compile policies efficiently?
• Relationships to the other papers we’ve read (e.g., HSA, VeriFlow, NICE, ndb)?
• Comparison of Pyretic and Maple?
• Support for distributed controllers, fault tolerance, supporting more sophisticated switches, etc.?