Software Defined Networking (SDN) Overview

02/08/2018

Sumit Maheshwari

Includes material from Scott Shenker and Nick McKeown
Outline

• Overview
  • What is SDN and why SDN
• SDN control plane abstractions
• Example applications
What is Software Defined Networking

A network in which the **control plane is physically separate from the forwarding plane**
and

**A single control plane** controls several forwarding devices
A Short History of SDN

• ~2004: Research on new management paradigms
  • RCP, 4D [Princeton, CMU,.....]
  • SANE, Ethane [Stanford/Berkeley]

• 2008: Software-Defined Networking (SDN)
  • NOX Network Operating System [Nicira]
  • OpenFlow switch interface [Stanford/Nicira]

• 2011: Open Networking Foundation (~69 members)
  • Board: Google, Yahoo, Verizon, DT, Microsoft, Facebook, NTT
  • Members: Cisco, Juniper, HP, Dell, Broadcom, IBM,.....

• 2013: Open Networking Summit
  • 1600 attendees, Google: SDN used for their WAN
  • Commercialized, in production use (few places)

• 2018: Yearly Summits and more Innovations
  • SDN conferences
  • Industry consortiums, ONF support, ONOS
Why SDN?

• **Networks are hard to manage**
  - Computation and storage have been virtualized
    - Creating a more flexible and manageable infrastructure
  - Networks are still notoriously hard to manage
    - Network administrators large share of sysadmin staff

• **Networks are hard to evolve**
  - Ongoing innovation in systems software
    - New languages, operating systems, etc.
  - Networks are stuck in the past
    - Routing algorithms change very slowly
    - Network management extremely primitive
Why SDN?

• Networks design not based on **formal principles**
  - OS courses teach fundamental principles
    - Mutual exclusion and other synchronization primitives
    - Files, file systems, threads, and other building blocks
  - Networking courses teach a big bag of protocols
    - No formal principles, just general design guidelines
Optimal Load Balancer: Ideally each HTTP request would be sent over a path which is lightly loaded to a server which is lightly loaded in order to minimize the request.
Optimal Load Balancer (Traditional)

It can choose only the lightly loaded server
The Network Control Problem

How does classical (IP-based) distributed network control plane operate today?

• Compute the configuration of each physical device
e.g., routing/forwarding tables, ACLs,…

• Operate without communication guarantees
e.g., best effort traffic

• Operate within given network-level protocol
e.g., different protocols for different use-cases

Only people who love complexity would find this a reasonable request
Idea: An OS for the Network
Idea: An OS for the Network

Control Programs

Network Operating System

- Simple Packet Forwarding Hardware
- Simple Packet Forwarding Hardware
- Simple Packet Forwarding Hardware
- Simple Packet Forwarding Hardware
Idea: An OS for the Network

**Software-Defined Networking (SDN)**

- Control Programs
- Global Network View
- Network Operating System

Control via forwarding interface

Protocols
Simplifying Control

Logically-centralized control

Smart & slow

API to the data plane (e.g., OpenFlow)

Dumb & fast

Switches
Outcome of SDN

• No longer designing distributed control protocols

• Much easier to write, verify, maintain, ...
  • an interface for programming

• Network OS (NOS) serves as fundamental control block
  • with a global view of network
Abstractions -> Problem Decomposition

- Decompose problem into basic components (tasks)
- Define an abstraction for each component
- Implementation of abstraction can focus on one task
- If tasks still too hard to implement, return to step 1

Problem: Find ‘x’
From Requirements to Abstractions

1. Operate without communication guarantees
   Need an abstraction for **distributed state**

2. Compute the configuration of each physical device
   Need an abstraction that **simplifies configuration**

3. Operate within given network-level protocol
   Need an abstraction for general **forwarding model**

*Once these abstractions are in place, control mechanism has a much easier job! Hopefully!*
Traditional Control Mechanisms

Distributed algorithm running between neighbors

Network of switches and/or routers
Restructured Network using SDN

1. Open interface to packet forwarding
2. At least one Network OS
   probably many.
   Open- and closed-source
3. Well-defined open API

Network OS

Packet Forwarding

Packet Forwarding

Packet Forwarding

Packet Forwarding

Packet Forwarding
How to design a Network Operating System?

- What features or “abstractions” should be provided by this “Network Operating System”?
- What should be the “global network view” & “programmatic interfaces” provided to control apps?
  - or what “low-level” details should be handled by Network OS?
- What is the granularity of control allowed to “apps”?
Requirements for these abstractions: Match-Action

**Plumbing Primitives:** `<Match, Action>`  

<table>
<thead>
<tr>
<th>Header</th>
<th>Data</th>
</tr>
</thead>
</table>

Match: `1000x01xx0101001x`

**Match** arbitrary bits in headers:
- Match on any header, or new header
- Allows any flow granularity

**Action**
- Forward to port(s), drop, send to controller
- Overwrite header with mask, push or pop
- Forward at specific bit-rate
Requirements for these abstractions: Flow based Decisions

“If header = x, send to port 4”
“If header = y, overwrite header with z, send to ports 5,6”
“If header = ?, send to me”

Traffic flow, packet flow or network flow

Sequence of packets from a source to a destination
(another host, a multicast group, or a broadcast domain)

An artificial logical equivalent to a call or connection
Example: IP Firewall (Reactive)

<table>
<thead>
<tr>
<th>SRC_IP=10.2.X.X</th>
<th>Forward to Port 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRC_MAC=D1.2E.X.X.X.X</td>
<td>BLOCK</td>
</tr>
</tbody>
</table>

Flow Table

<table>
<thead>
<tr>
<th>Match</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>NULL</td>
</tr>
</tbody>
</table>

---
Example: IP Firewall (Reactive)

<table>
<thead>
<tr>
<th>SRC_MAC=01.2E.X.X.X</th>
<th>BLOCK</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.2.1.19</td>
<td>Forward to Port 2</td>
</tr>
<tr>
<td>SRC_IP=10.2.X.X</td>
<td></td>
</tr>
</tbody>
</table>

Flow Table
Example: IP Firewall (Reactive)

<table>
<thead>
<tr>
<th>SRC_MAC=01.2E.X.X.X.X</th>
<th>BLOCK</th>
</tr>
</thead>
</table>

**Flow Table**

<table>
<thead>
<tr>
<th>Match</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRC_IP=10.2.X.X</td>
<td>Forward to Port 2</td>
</tr>
</tbody>
</table>

10.2.1.19
Example: Firewall/Switching (Proactive)

<table>
<thead>
<tr>
<th>SRC_MAC</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1.2E.X.X.X.X</td>
<td>BLOCK</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SRC_IP</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.2.X.X.X</td>
<td>Forward to Port 2</td>
</tr>
</tbody>
</table>

Flow Table

<table>
<thead>
<tr>
<th>Match</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRC_IP=10.2.X.X</td>
<td>Forward to Port 2</td>
</tr>
</tbody>
</table>
Example: Firewall/Switching (Proactive)

Flow Table

<table>
<thead>
<tr>
<th>Match</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRC_IP=10.2.X.X</td>
<td>Forward to Port 2</td>
</tr>
<tr>
<td>SRC_MAC=D1.2E.X.X.X.X</td>
<td>BLOCK</td>
</tr>
</tbody>
</table>
Examples Syntax from OpenFlow

### Switching

<table>
<thead>
<tr>
<th>Switch Port</th>
<th>MAC src</th>
<th>MAC dst</th>
<th>Eth type</th>
<th>VLAN ID</th>
<th>IP Src</th>
<th>IP Dst</th>
<th>IP Prot</th>
<th>TCP sport</th>
<th>TCP dport</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
<td>00:1f...</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>port6</td>
</tr>
</tbody>
</table>

### Routing

<table>
<thead>
<tr>
<th>Switch Port</th>
<th>MAC src</th>
<th>MAC dst</th>
<th>Eth type</th>
<th>VLAN ID</th>
<th>IP Src</th>
<th>IP Dst</th>
<th>IP Prot</th>
<th>TCP sport</th>
<th>TCP dport</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>5.6.7.8</td>
<td>*</td>
<td>port6</td>
</tr>
</tbody>
</table>

### Firewall

<table>
<thead>
<tr>
<th>Switch Port</th>
<th>MAC src</th>
<th>MAC dst</th>
<th>Eth type</th>
<th>VLAN ID</th>
<th>IP Src</th>
<th>IP Dst</th>
<th>IP Prot</th>
<th>TCP sport</th>
<th>TCP dport</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>22</td>
<td>drop</td>
</tr>
</tbody>
</table>
SDN Summary

• General **forwarding model** (data plane abstraction)
  • Currently based on OpenFlow (flow-level) forwarding model
    • *prioritized* rules [header: counters, actions]: match → actions
  • assume forwarding elements provide (standardized) APIs
    • install and manipulate forwarding tables, perform match and actions, & collect stats, etc.

• Logically centralized control plane (a “network OS”)
  • serve as a “network operating system”
    • provide distributed state management, map control logic to data plane actions, etc.
  • provide a “**global network view**” to (high-level) “control apps”
    • enable “higher-level” abstractions to hide “lower-level” details

• Control apps operate on higher-level **abstractions**
  • control apps focus on “control logic” using network OS APIs
  • *Hopefully, much easier to write, verify and maintain!*