Introduction to Software-Defined Networking
UG3 Computer Communications & Networks (COMN)

Myungjin Lee
myungjin.lee@ed.ac.uk

Courtesy note: Slides from course CPS514 Spring 2013 at Duke University and Hot Interconnects Keynote by Nick McKeown, 2012
Outline

• What is SDN?
  – Limitations of current networks
  – The idea of Network OS

• What is OpenFlow?
  – How it helps SDN

• Application: Network debugging with SDN
Limitations of Current Networks
Limitations of Current Networks

- Enterprise networks are difficult to manage

- “New control requirements have arisen”:
  - Greater scale
  - Migration of VMs

- How to easily configure huge networks?
Limitations of Current Networks

- Old ways to configure a network
Limitations of Current Networks

- Million of lines of source code
- Billions of gates
- Many complex functions baked into infrastructure
  - OSPF, BGP, multicast, differentiated services, Traffic Engineering, NAT, firewalls, ...

Cannot dynamically change according to network conditions
Limitations of Current Networks

• No control plane abstraction for the whole network!

• It’s like old times – when there was no OS…

Wilkes with the EDSAC, 1949
Idea: An OS for Networks

OpenFlow/SDN tutorial, Srini Seetharaman, Deutsche Telekom, Silicon Valley Innovation Center
Idea: An OS for Networks

Control Programs

Network Operating System

Specialized Packet Forwarding Hardware

Operating System

OpenFlow/SDN tutorial, Srini Seetharaman, Deutsche Telekom, Silicon Valley Innovation Center
Idea: An OS for Networks

Control Programs

Network Operating System

Simple Packet Forwarding Hardware

Simple Packet Forwarding Hardware

Simple Packet Forwarding Hardware

Simple Packet Forwarding Hardware

OpenFlow/SDN tutorial, Srini Seetharaman, Deutsche Telekom, Silicon Valley Innovation Center
Idea: An OS for Networks

• “NOX: Towards an Operating System for Networks”

Software-Defined Networking (SDN)
Software Defined Networking

- No longer designing distributed control protocols

- Much easier to write, verify, maintain, …
  - An interface for programming

- NOS serves as fundamental control block
  - With a global view of network
Software Defined Networking

• Questions:
  – How to obtain global information?
  – What are the configurations?
  – How to implement?
  – How is the scalability?
  – How does it really work?
Outline

• What is SDN?
  – Limitations of current networks
  – The idea of Network OS

• What is OpenFlow?
  – How it helps SDN

• Application: Network debugging with SDN
OpenFlow

• “OpenFlow: Enabling Innovation in Campus Networks”

• Like hardware drivers
  – interface between switches and Network OS
Getting Started

OpenFlow Tutorial

- search: “OpenFlow Tutorial”

Mininet

- Network emulator
- Designed for emulating SDN networks
- Easy to use
- High performance (100 nodes on a laptop)
- search: “Mininet”
OpenFlow Switches?

Software switch
- Open vSwitch (openvswitch.org)
- Now part of Linux distribution

Hardware switches
- Announcements from several vendors
- HP, Brocade, NEC, …
- (You could ask Google for one of theirs 😊)
OpenFlow Basics
OpenFlow Basics

OpenFlow Controller

OpenFlow Protocol (SSL)

Flow table
1. <Rule1, port1>
2. <Rule2, drop>
3. <Rule3, port2>
4. <Rule4, port2>

Data Path (Hardware)

Control Path

No match
OpenFlow Switching

OpenFlow Client

OpenFlow Table

<table>
<thead>
<tr>
<th>MAC src</th>
<th>MAC dst</th>
<th>IP Src</th>
<th>IP Dst</th>
<th>TCP sport</th>
<th>TCP dport</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>*</td>
<td>*</td>
<td>5.6.7.8</td>
<td>*</td>
<td>*</td>
<td>port 1</td>
</tr>
</tbody>
</table>

Controller

PC

Software Layer

Hardware Layer

OpenFlow Table Entry

<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>
</tr>
</thead>
</table>

+ mask

Packet + byte counters

1. Forward packet to port(s)
2. Encapsulate and forward to controller
3. Drop packet
4. Send to normal processing pipeline
5. ….

## OpenFlow Examples

### 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></td>
<td>00:1f:..</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>*</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>*</td>
<td>22</td>
</tr>
</tbody>
</table>
OpenFlow Usage

OpenFlow/SDN tutorial, Srini Seetharaman, Deutsche Telekom, Silicon Valley Innovation Center
Alice’s code:
> Simple learning switch
> Per Flow switching
> Network access control/firewall
> Static “VLANs”
> Her own new routing protocol: unicast, multicast, multipath
> Home network manager
> Packet processor (in controller)
> IPvAlice
OpenFlow

- Standard way to control flow-tables in commercial switches and routers
- Just need to update firmware
- Essential to the implementation of SDN
Centralized/Distributed Control

• “Onix: A Distributed Control Platform for Large-scale Production Networks”
Ongoing focuses of SDN

- New policies for security
- Programmable WLANs
- The placement of controllers (amount; location; centralized/distributed)
- Debugger for SDN
Outline

• What is SDN?
  – Limitations of current networks
  – The idea of Network OS

• What is OpenFlow?
  – How it helps SDN

• Application: Network debugging with SDN
Making Networks Work (Today)

traceroute, ping, tcpdump, SNMP, Netflow

.... er, that’s about it.
Why debugging networks is hard

Complex interaction
- Between multiple protocols on a switch/router.
- Between state on different switches/routers.

Multiple uncoordinated writers of state.

Operators can’t...
- Observe all state.
- Control all state.
Networks are kept working by “Masters of Complexity”

A handful of books
Almost no papers
No classes

Many papers since 2012 in the SDN context
Philosophy of Making Networks Work

YoYo
“You’re On Your Own”

Yo-Yo Ma
“You’re On Your Own, Mate”
With SDN we can:

1. Formally verify that our networks are behaving correctly
2. Identify bugs, then systematically track down their root cause
Software Defined Network (SDN)

```
firewall.c
...
if ( pkt->tcp->dport == 22)
  dropPacket(pkt);
...
```

1. <Match, Action>
2. <Match, Action>
3. <Match, Action>
4. <Match, Action>
5. <Match, Action>
6. ...
7. ...

Packet Forwarding

Control Programs

Abstract Network View

Network Virtualization

Global Network View
Example debugging tools

1. Interactive Debugging [ndb]
   “Finding bugs, and their root cause, in an operational network”

2. PathDump: A path-tracing based debugger
   “Tracing paths of individual packets and debugging problems in a datacenter network”
1. Interactive Debugging
Finding bugs, and their root cause, in an operational network

Nikhil Handigol
Brandon Heller
Vimal Jeyakumar
David Mazières

Stanford University
Function A():
   \( u = B(v) \)

Function B():
   \( w = C(x) \)

Function C():
   \( y = \text{error} \)

Breakpoint
"u == error"

Backtrace:
File “A”, line 10, Function A()
File “B”, line 43, Function B()
File “C”, line 21, Function C()
Interactive Debugging with \textit{ndb}

Problem

When an operational network misbehaves, it is very hard to find the root cause

Goal

– Allow users to define a \underline{Network Breakpoint}
– Capture and reconstruct the sequence of events leading to the breakpoint.
Network Debugger

Breakpoint
Switch = S
IP src = A, IP dst = B
TCP Port = 22
Network Debugger

Stanford Backbone

1. <Match, Action>
2. <Match, Action>
3. <Match, Action>
4. <Match, Action>
5. <Match, Action>
6. ...
7. ...

Collector
Network Virtualization

Network OS

Control Programs

3. Port == 22, Drop

Breakpoint

Switch = S
IP src = A, IP dst = B
TCP Port = 22

Collector
1. `<Match, Action>`
2. `<Match, Action>`
3. `<Port == 22, Drop>`
4. `<Match, Action>`
5. `<Match, Action>`
6. `<Match, Action>`

Breakpoint

Switch = S
IP src = A, IP dst = B
TCP Port = 22

```
firewall.c
...
if ( pkt->tcp->dport == 22)
  dropPacket(pkt);
...
```
Who benefits

Network developers
  – Programmers debugging control programs

Network operators
  – Find policy error
  – Send error report to switch vendor
  – Send error report to control program vendor
2. PathDump

Tracing paths of individual packets and debugging problems in a datacenter network

Packet trajectory tracing

• “Tracing” the path taken by the packet
  – Scalability: Switch flow rules, packet header space
• Policy: **All packets from 1 to 3 must avoid 4**

![Diagram of network topology with nodes 1, 2, 3, 4, 5, 6 and arrows indicating packet flow. The actual path is from 1 → 4 → 3 with a violation indicated.]
Packet trajectory tracing

• Checks whether packet followed route as defined in policy
• Helps to localize network problems
  Ex: misconfiguration, failures
PathDump architecture

1. Switch embeds unique ID (e.g., link ID)
PathDump architecture

- Packet header space limitation
- **Cherrypick** [SOSR’15] for current deployments
PathDump architecture

1. Switch embeds unique ID (e.g., link ID)

Only one shortest path from Core to Dst
PathDump architecture

2. End-host captures packet path and updates flow-level statistics
PathDump architecture

2. End-host captures packet path and updates flow-level statistics
PathDump architecture

3. Aggregator runs debugging applications

On-demand vs. Event-driven
PathDump interface

A small set of simple APIs enables a variety of debugging applications

- Other end-host APIs: getCount(), getPoorTCPFlows(), Alarm(), etc.
- Aggregator APIs: Install(), execute() and uninstall()
Example 1: Path conformance

- Check if actual forwarding path $\neq$ network policy
  - May occur due to switch faults or network state change

Policy: Packet must avoid switch 4

Actual Path: 1 – 4 – 3
Example 1: Path conformance

- Check if actual forwarding path $\neq$ network policy
  - May occur due to switch faults or network state change

```python
# Given flowID, paths, switchID
1: for path in paths:
2:   if switchID in path:
3:     Alarm(flowID, PC_FAIL, result)
```
Example 2: Silent random packet drop diagnosis

1. Install(query)
2. Alarm()
3. getPaths()
4. Max-Coverage algorithm
   - A – B : 2
   - B – C : 1
   - B – D : 1

• No packet drop hint
• Software/Hardware bug