Floodlight and the OpenSDN Stack

Rob Sherwood – Big Switch Networks
Joseph Tardo – Broadcom
Douglas Flint – Broadcom
Example SDN Stack: Derived from ONS Logo

- SDN App1
- SDN App2
- Controller Platform
- OF Driver
- OpenFlow Agent
- Linux OS
- ASIC SDK
- CPU
- ASIC

SDN Controller
This workshop: Top-to-Bottom Open SDN Stack

SDN App1
SDN App2

Controller Platform

OF Driver

OpenFlow Agent

Linux OS
ASIC SDK

CPU
ASIC

Floodlight SDN Controller

LOXIGEN – LOCI, OpenFlowJ

Indigo OpenFlow Agent

NEW! OF-Datapath Abstraction (OF-DPA)

NEW! Open Network Linux
<table>
<thead>
<tr>
<th>Time</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>8:30-9:00a</td>
<td>Open SDN Stack Overview</td>
</tr>
<tr>
<td>9:00-10:00a</td>
<td>Open Source Switch OS: Open Network Linux</td>
</tr>
<tr>
<td>10:00-11:30a</td>
<td>OF-DPA: Open API for Broadcom StrataXGS Architecture</td>
</tr>
<tr>
<td>11:30-12:30p</td>
<td>Lunch</td>
</tr>
<tr>
<td>12:30-1:00p</td>
<td>Indigo OpenFlow 1.3 Architecture</td>
</tr>
<tr>
<td>1:00-2:00p</td>
<td>Floodlight, LOXI, and OpenFlow 1.3 Support</td>
</tr>
</tbody>
</table>
Overview Outline: 8:30-9:00a

• Introductions
• Open SDN Stack Component Summary

<table>
<thead>
<tr>
<th>Components</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware Design</td>
<td>OpenFlow Agent</td>
</tr>
<tr>
<td>Operating System</td>
<td>OpenFlow Protocol Driver</td>
</tr>
<tr>
<td>Hardware SDK</td>
<td>SDN Controller</td>
</tr>
</tbody>
</table>

• OpenFlow – the short, short version
  • For detailed introduction to OpenFlow, see other tutorials
Introductions: Who Are We?

• Rob Sherwood
  • CTO at Big Switch Networks

• Joseph Tardo
  • Associate Technical Director, OpenFlow and SDN technologies, Broadcom

• Douglas Flint
  • Principal Engineer - Software Systems, Broadcom
Introductions: Who Are You?

By a show of hands...

• Developers with Floodlight or Indigo experience?
• Developers with Other SDN experience?
• Developers with Networking Experience?
• Developer with Other Experience?
• Technical, but not a developer?
• Non-technical?
Open Hardware

• Open Compute Project (OCP) hosts open hardware designs
  • [http://opencompute.org/network](http://opencompute.org/network)

• Hardware designs:
  • Schematics, board layout
  • Recommended devices
  • Port counts, fans, power requirements

<table>
<thead>
<tr>
<th>Components</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Hardware Design</strong></td>
<td>OpenFlow Agent</td>
</tr>
<tr>
<td>Operating System</td>
<td>OpenFlow Protocol Driver</td>
</tr>
<tr>
<td>Hardware SDK</td>
<td>SDN Controller</td>
</tr>
</tbody>
</table>
Open Operating System

- Open Network Linux: Linux distribution for network devices
  - http://github.com/opennetworklinux/ONL
- Based on Debian Wheezy, but adds:
  - Open Network Install Environment (ONIE) compatibility
  - Lots of switch-specific device drivers: I2C, GPIO, device tree files for non-x86
  - Niceties for switches: network booting, overlayfs over flash
  - Support for increasing number of network devices

<table>
<thead>
<tr>
<th>Components</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware Design</td>
<td>OpenFlow Agent</td>
</tr>
<tr>
<td><strong>Operating System</strong></td>
<td>OpenFlow Protocol Driver</td>
</tr>
<tr>
<td>Hardware SDK</td>
<td>SDN Controller</td>
</tr>
</tbody>
</table>
Hardware SDK

• OpenFlow Datapath Abstraction
  • TBD – press release today!
• From Documentation:
  • “OpenFlow Data Plane Abstraction (OF-DPA) is an application software component that implements an adaptation layer between OpenFlow and the Broadcom Silicon SDK. OF-DPA enables scalable implementation of OpenFlow 1.3 on Broadcom switch devices.”

<table>
<thead>
<tr>
<th>Components</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware Design</td>
<td>OpenFlow Agent</td>
</tr>
<tr>
<td>Operating System</td>
<td>OpenFlow Protocol Driver</td>
</tr>
<tr>
<td><strong>Hardware SDK</strong></td>
<td>SDN Controller</td>
</tr>
</tbody>
</table>
OpenFlow Agent

• Indigo OpenFlow Agent
  • http://projectfloodlight.org/indigo
• Supports:
  • OpenFlow 1.0 and OpenFlow 1.3
  • x86-based vSwitch: http://www.projectfloodlight.org/indigo-virtual-switch/
  • Broadcom-based Hardware switches, including OF-DPA
• Uses LOXI for its OpenFlow driver

<table>
<thead>
<tr>
<th>Components</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware Design</td>
<td>OpenFlow Agent</td>
</tr>
<tr>
<td>Operating System</td>
<td>OpenFlow Protocol Driver</td>
</tr>
<tr>
<td>Hardware SDK</td>
<td>SDN Controller</td>
</tr>
</tbody>
</table>
OpenFlow Protocol Driver

- **LoxiGen**: Generates OF language specific protocol bindings
  - [http://github.com/floodlight/loxigen](http://github.com/floodlight/loxigen)
- Generates OpenFlow v1.0-v1.3.1+ bindings for:
  - C – for Indigo
  - Java – for Floodlight
  - Python – for OFTest: [https://github.com/floodlight/oftest](https://github.com/floodlight/oftest)
  - Wireshark/Lua: [http://www.projectfloodlight.org/openflow.lua](http://www.projectfloodlight.org/openflow.lua)

<table>
<thead>
<tr>
<th>Components</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware Design</td>
<td>OpenFlow Agent</td>
</tr>
<tr>
<td>Operating System</td>
<td><strong>OpenFlow Protocol Driver</strong></td>
</tr>
<tr>
<td>Hardware SDK</td>
<td>SDN Controller</td>
</tr>
</tbody>
</table>
SDN Controller

- **Floodlight SDN Controller**
  - [http://projectfloodlight.org](http://projectfloodlight.org)

- **Supports:**
  - OpenFlow 1.0 \(\rightarrow\) TODO: Update to OpenFlow 1.3/Loxigen/OpenFlowJ
  - Example apps: learning switch, hub, skeleton OpenStack Neutron support
  - Last year: example use cases from CERN/Caltech and Mellanox/Radware
  - This year: many of the Research Track papers implement on Floodlight

<table>
<thead>
<tr>
<th>Components</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware Design</td>
<td>OpenFlow Agent</td>
</tr>
<tr>
<td>Operating System</td>
<td>OpenFlow Protocol Driver</td>
</tr>
<tr>
<td>Hardware SDK</td>
<td><strong>SDN Controller</strong></td>
</tr>
</tbody>
</table>
OpenFlow – the short, short version
EXISTING NETWORK DEVICES – “CLOSEDFLOW”

Exact Same Process for:

- "128.8.0.0/16"
- "Closed Flow"

BGP
MPLS LDP
OSPF/ISIS
STP

Exact Same Process for:

- "128.8.0.0/16"
- "Closed Flow"

BGP
MPLS LDP
OSPF/ISIS
STP
OPENFLOW IS REMOTE CONTROL PLANE PROTOCOL

Just like “closed flow”, but over network and open!
OPENFLOW IN PRACTICE

Sequence of tables in a packet processing pipeline

OpenFlow Switch

Flow Table

OF Agent

OpenFlow Protocol on SSL

Server

OpenFlow Controller

Linux

Management Network

OpenFlow Switch

OpenFlow Switch

OpenFlow Switch

OpenFlow Switch
### Flow Table Abstraction

**Sequence of tables in a packet processing pipeline**

<table>
<thead>
<tr>
<th>Priority</th>
<th>Match</th>
<th>Action List</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>IP.proto=6</td>
<td>TTL--, Fwd:port 3</td>
</tr>
<tr>
<td></td>
<td>TCP.dst=22</td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>IP.dst=128.8/16</td>
<td>Queue: 4</td>
</tr>
<tr>
<td>100</td>
<td>*</td>
<td>DROP</td>
</tr>
</tbody>
</table>

- Existing networking hardware *actually very flexible*
- **FUD**: Large + narrow versus small + wide match tables
- Active work in the Open Networking Foundation to bring OpenFlow to feature parity with “closed flow”
DECOUPLE CONTROL FROM FORWARDING

- OpenFlow permits fewer controllers than datapaths.
- Reduced number of management touchpoints.
- Mapping from datapaths to controllers is a crucial network design question.

OpenFlow does not imply a single point of failure!

Allows load balancing and failover.
• Introductions
• Open SDN Stack Component Summary

<table>
<thead>
<tr>
<th>Components</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware Design</td>
<td>OpenFlow Agent</td>
</tr>
<tr>
<td>Operating System</td>
<td>OpenFlow Protocol Driver</td>
</tr>
<tr>
<td>Hardware SDK</td>
<td>SDN Controller</td>
</tr>
</tbody>
</table>

• OpenFlow – the short, short version
  • For detailed introduction to OpenFlow, see other tutorials
<table>
<thead>
<tr>
<th>Time</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>8:30-9:00a</td>
<td>Open SDN Stack Overview</td>
</tr>
<tr>
<td>9:00-10:00a</td>
<td>Open Source Switch OS: Open Network Linux</td>
</tr>
<tr>
<td>10:00-11:30a</td>
<td>OF-DPA: Open API for Broadcom StrataXGS Architecture</td>
</tr>
<tr>
<td>11:30-12:30p</td>
<td>Lunch</td>
</tr>
<tr>
<td>12:30-1:00p</td>
<td>Indigo OpenFlow 1.3 Architecture</td>
</tr>
<tr>
<td>1:00-2:00p</td>
<td>Floodlight, LOXI, and OpenFlow 1.3 Support</td>
</tr>
</tbody>
</table>
Open Network Linux

A Common Linux Platform for Open Network Devices

Rob Sherwood
Big Switch Networks
CTO
Open Network Linux: Outline

History

Motivation
- Elements of a Linux Distribution
- Tower of Babel
- Benefits for users, vendors

Technical details
- Multi-platform support: x86, PPC, and x86 VM
- Full “Server-like” experience on network hardware
- Network booting and image management
OPEN NETWORK LINUX (ONL) HISTORY

• Switch Light is a Big Switch Networks commercial product
  • Switch Light = Indigo + ONL
• Proposed to open source the Linux bits of Switch Light to OCP
  • November OCP Network meeting
  • Lots of good community feedback
• Source code went live for OCP 1/27/2014 Summit
• Work in progress: github.com/opennetworklinux/ONL
  • Needs website, pre-compiled binaries
  • Lots of interest from the ODM community
BACKGROUND – DISTRIBUTION

“Linux” proper is just the kernel

• “Distribution” is everything else
  • e.g., RedHat, Ubuntu, Slackware, Gentoo, etc.
  • Libc, compiler, user space binaries
  • Configuration, file system layout, startup scripts
  • Package management, full-featured boot loader

• A lot of work goes into making a good distribution
  • Default configurations, daemons
  • Q/A (lots and lots)

• Lots of possibilities for niche distributions
  • e.g., embedded environments differ from server
  • Bootloaders vs. full-fledged systems
BACKGROUND – PLATFORM SUPPORT
Sometimes we forget, but the boot process is horrible

- **Server and switch platforms have many idiosyncrasies**
  - Litany of little devices we never think of...
  - USB, GPIO, flash, PCI, serial, RTC, EEPROM, DMA, Crypto chips
  - MPIC - Multiple programmable interrupt controller
  - ... all at platform-specific memory locations

- **x86-based standards shield us from low-level platform details**
  - Vendor must write a BIOS for each platform, e.g., ACPI standard
  - Operating systems (e.g., Linux) discovery devices via BIOS

- **But switch platform ecosystem is not as evolved**
  - Includes switch specific devices, like I2C, GPIOs, etc.
  - Manual map/inventory of hardware ➔ memory address via Device Tree Source (DTS) files
Motivation: Tower of Babel is Bad

Stack #1
- STP + MLAG
- Fedora Linux Kernel
- Device Tree #1
- Initrd #1
- OCP Platform V1

Stack #2
- OpenFlow daemon
- Std. Debian Linux Kernel
- Device Tree #2
- Initrd #2
- OCP Platform V2

Stack #3
- Quagga + hooks
- BusyBox Linux Kernel
- Device Tree #3
- Initrd #3
- White box vendor

Switch Agent(s)
Platform
- Independent
- Dependent
- Hardware Layer
Proposal: Common Linux Platform

- **STP + MLAG**
- **OpenFlow daemon**
- **Quagga + hooks**

**Unified Device Tree Repository**

**Unified Driver Repository**

**Standard packages, tools, etc. Stock Linux Kernel + any patches**

**OCP Platform V1**

**OCP Platform V2**

**White box vendor**

- Keep differentiation in switch agents
- Come together around the common bits
- Maximize hardware abstraction
Benefits for Users

• Help foster an OSS ecosystem sandbox
  
  Easy OS binary to download and play with

• Vendor agnostic common Linux platform
  
  Deploy your non-switch tools on any box
  
  e.g., Chef/puppet/custom
  
  Manage the switch like any other server

• Central repository for DTS files
  
  Less friction to support new platforms
  
  Easy hardware validation
Benefits for vendors

• All vendors already have their own distributions
  
  Informal check: most are based off of Debian Wheezy
  
  No significant space for differentiation, might as well standardize
  
  Reduce engineering effort

• Reduce the effort to support new platforms
  
  Open up the ecosystem – good for everyone
  
  Central repository for hardware vendors to test their drivers

• Normalize hardware compatibility lists
Open Network Linux: **Goals**

**Accelerate adoption of OCP and other switch hardware**

- Users: download image, install via ONIE
- Vendors: common Linux platform for new drivers, testing

**Create an open community**

- Target: Linux portable to all networking devices

**License: Eclipse Public License and GPL for Kernel**

**“What’s in it for me?”**

- Engineering efficiencies
- Better development and deployment experience
Lots of support from community – thanks!

Github went live: 1/27/2014

Main repository: github.com/opennetworklinux/ONL

Builds ONIE-compatible images for:

Generic x86 platforms: Interface Masters not *yet* tested
Many PPC platforms: Quanta LY2, LB9, LB8D, Accton 5652
x86 VM build: for testing – qcow2 (vmdk via convert)

Stability level: “works for us”

Feedback welcome
Open Network Linux: Outline

History

Motivation

- Elements of a Linux Distribution
- Tower of Babel
- Benefits for users, vendors

Technical details

- Multi-platform support: x86, PPC, and x86 VM
- Full “Server-like” experience on network hardware
- Network booting and image management
Technical Overview

Code builds two main artifacts:

- **ONL installer/loader**: like grub, but multi-platform with netboot
- **ONL SWI file**: zip’d Switch Image with root fs, kernel, initrd

Code is divided into multiple sub-modules

- **ONL**: main repository – auto pulls in other repos
- **linux-3.9.6**: extracted kernel code
- **loader**: scripts and code for boot loader process
- **infra and common**: libraries, shared routines, faultd
Quick Aside: Open Network Install Environment (ONIE)

• Open source project to install/uninstall network OS

• Think of it like a hybrid PC BIOS and Grub/LILO/Sysimage

• Co-operative project: OCP, Cumulus, Big Switch, Others
  • In practice: Curt Brune from Cumulus Networks does almost all of the work

• Allows a network admin to install/uninstall a network OS
  • In practice, it is itself a ~4MB mini-Linux installation
  • More details later
Deployment Overview

Full documentation in README in ONL repository

1. Get code from github.com/opennetworklinux/ONL
2. (only for ppc) Build a cross-compilation workspace
3. Build ONL installer/loader image
4. Put ONL installer/loader image on ONIE server
5. Boot switch and install ONL via ONIE
6. Build one or more ONL SWI’s
7. Netboot from scp/nfs/http/ftp/etc. to install ONL SWI
ONL is Multi-Platform

Support many boxes from the same code-base

Open Network Linux:
- Kernel
- Drivers
- Loader
- Work flow
- Build scripts
- Management Model

X86 Arch
- Interface Master's
- x86 VM
- others?

PPC
- Quanta LB9, LY2, LY5
- Accton 5652
- Delta, Alpha, etc.

ARM?
- ???
Install Using ONIE then Boot ONL

Boot Logic:

1. uBoot POSTs
2. $nos_boot_cmd is read from ENVs
3. run $nos_boot_cmd
   • If $nos_boot_cmd returns, run ONIE
   • On install, ONIE sets $nos_boot_cmd to load ONL loader
4. Loader downloads specified SWI URL if not cached
5. Loader mounts rootfs as ramdisk with overlayfs
6. ONL loader kexec’s SWI kernel

~64MB

- uBoot
- ENVs
- ONIE
- Free Space

~2GB

- ONL Loader
- ONL SWI #1 (cached)
- ONL SWI #2 (cached)

Mass Storage
PERSPECTIVE RELATIVE TO ONIE
Different tools for different use cases – but maximize code reuse

- ONIE First boot Loader ~3MB
- Normal Full-featured Boot Loader (w/BusyBox) ~16MB
- Main Network OS Image (.swi) (w/real binaries) ~160 MB

Github.com/onie/onie

Proposed OCP Distribution

Common kernel and DTS files??
Tricks to Use Switches Like Servers

Switches have flash, not hard drives

Problem 1: Maximum flash cycle time limit disk writes
Problem 2: Flash and ram more limited than typical servers
Fix: Use overlayfs to overlay copy-on-write ram disk over flash

ONL uses full-featured binaries

For size, most switch OS’s use stripped binaries, e.g., busybox
Bigger binaries uses additional space, but ok with overlayfs
Install/use proper Debian binaries using apt-get
Useful for development or operations, e.g., gcc or Chef/Puppet
ONL Supports Net Booting Natively

Boot SWIs over the network once ONL Loader installed

- SWIs are cached locally for performance, resilience
- Simplifies operational management, upgrades
- Supports http, ftp, tftp, nfs, ssh/scp, or ZTN

Zero-touch Networking (ZTN): auto-discover SWIs

- Like PXE for your switches
- Query all http servers in local subnet, use SWI of first hit
- Just like ONIE

Netboot makes managing your network much easier
Outline/Schedule: Working Bottom-Up

<table>
<thead>
<tr>
<th>Time</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>8:30-9:00a</td>
<td>Open SDN Stack Overview</td>
</tr>
<tr>
<td>9:00-10:00a</td>
<td>Open Source Switch OS: Open Network Linux</td>
</tr>
<tr>
<td>10:00-11:30a</td>
<td>OF-DPA: Open API for Broadcom StrataXGS Architecture</td>
</tr>
<tr>
<td>11:30-12:30p</td>
<td>Lunch</td>
</tr>
<tr>
<td>12:30-1:00p</td>
<td>Indigo OpenFlow 1.3 Architecture</td>
</tr>
<tr>
<td>1:00-2:00p</td>
<td>Floodlight, LOXI, and OpenFlow 1.3 Support</td>
</tr>
</tbody>
</table>
... hand over to Broadcom folks.
<table>
<thead>
<tr>
<th>Time</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>8:30-9:00a</td>
<td>Open SDN Stack Overview</td>
</tr>
<tr>
<td>9:00-10:00a</td>
<td>Open Source Switch OS: Open Network Linux</td>
</tr>
<tr>
<td>10:00-11:30a</td>
<td>OF-DPA: Open API for Broadcom StrataXGS Architecture</td>
</tr>
<tr>
<td>11:30-12:30p</td>
<td>Lunch</td>
</tr>
<tr>
<td>12:30-1:00p</td>
<td>Indigo OpenFlow 1.3 Architecture</td>
</tr>
<tr>
<td>1:00-2:00p</td>
<td>Floodlight, LOXI, and OpenFlow 1.3 Support</td>
</tr>
</tbody>
</table>
Indigo OpenFlow Agent

• Different from “Indigo v1”
  • Complete rewrite
  • Uses LoxiGen/Loci – not openflow.h

• Design Goal:
  • Highly portable across ASICs, platforms, operating systems
  • Top half/bottom half design: separate hardware dependent bits
  • Roughly Agnostic to OpenFlow version – work on intermediate form
  • Composed of lots and lots of modular libraries
Indigo “Bottom Half” Ports

- Broadcom SDK – Binary only
- Broadcom OF-DPA – source code with OF-DPA distribution
- Mellanox’s SDK – Demo only (?)
- And others...
Switch Light is our Indigo OpenFlow Agent running on Open Network Linux on x86 or Broadcom hardware.
## Outline/Schedule: Working Bottom-Up

<table>
<thead>
<tr>
<th>Time</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>8:30-9:00a</td>
<td>Open SDN Stack Overview</td>
</tr>
<tr>
<td>9:00-10:00a</td>
<td>Open Source Switch OS: Open Network Linux</td>
</tr>
<tr>
<td>10:00-11:30a</td>
<td>OF-DPA: Open API for Broadcom StrataXGS Architecture</td>
</tr>
<tr>
<td><strong>11:30-12:30p</strong></td>
<td>&lt;b&gt;Lunch&lt;/b&gt;</td>
</tr>
<tr>
<td>12:30-1:00p</td>
<td>Indigo OpenFlow 1.3 Architecture</td>
</tr>
<tr>
<td>1:00-2:00p</td>
<td>Floodlight, LOXI, and OpenFlow 1.3 Support</td>
</tr>
</tbody>
</table>
Challenge: Floodlight Needs OpenFlow 1.3 Support!

- All of the right bits are there
  - All of the language bindings are done
- Just need to update the handshaking and example code
  - A mere mater of days of work – “TM”
  - But I haven’t had the time
- Is this a possible community activity?
  - “openflow-1.3” branch on github.com/floodlight/floodlight
LOXI is fully OF1.3.1

git://github.com/floodlight/loxigen
OpenFlowJng: Architecture Ideas for OF 1.x compatibility
What's this about?

A new OpenFlow API for Floodlight with support for OF1.[0-3+]

- Message classes
- Message/wire conversions
- Types

How should this API look like?
It's not...

...about implementation details

Code generated by LOXI, but that's beside the point.
Higher Level Stuff: Not Today!

Portability:
abstract away OF
version / hardware
differences
virtualize resources

Network model:
a higher level / network
wide forwarding
forwarding abstraction

Hard problems!
Solutions to be built on top of this
Why?

• Current API has prob... err, optimization potential
  o mutable state passed around 100s lines of code
  o int bitmasks -> people use magic numbers
  o manual housekeeping of line format metadata
    o Forgot to update size, anyone?
  o messages don't guarantee invariants...

• A convenient/safe API helps
  o test coverage
  o correctness
Design Goals
Multiple Versions support

- OF 1.[0-3]+ compatibility
  - Easy upgrade path for newer versions
  - Support for Vendor extensions
  - Expose extended features
  - Provide "1.0 compatible" baseline
Bug Resilience

- Type safety (enums vs. short type)
- No Integer bitmasks
  - necessary for multi-version support
- No manual housekeeping of wireformat properties
- Localize state mutation / limit passing of mutable state
More...

- Convenience, also for manual creation
  - Unit tests!

- Enable Performance optimizations
  - Lazy parsing
  - Caching of messages
Design Goals - in a nutshell

"Work hard in the API to make life easy for apps/clients"
Architecture

(no UML[*]. Promised.)

[*] almost
Interfaces and Value Classes

**Interfaces**
All OF Message types exposed as Interfaces
(Messages, Actions, Matches etc.)

**Value Classes**
for constant, simple concepts (IPv4Address, MacAddress)

stuff that (potentially) changes between versions

stuff that always stays the same
Interfaces

• One Interface for all versions
  o provides Union of the features
  o supports querying for features
  o throws UnsupportedOperationException()

• version specific implementations
  o package private

<<interface>> OFFlowMod

OFFlowModImplVer10
OFFlowModImplVer11
OFFlowModImplVer12
Immutability

All Message / Value Objects are immutable(*)

How do you create stuff?

(*) externally visible
Instance creation

Instance controlled

No constructors. Ever.

Complex OF Message types:
Factories / Builders
factory.createBuilder()

Simple Value Objects:
Factory Methods
IPv4.of("1.2.3.4/24")

But I hate factories!!! ☠
Message Objects and Builders

- **Factory**
  - createBuilder()

- **Builder**
  - setFoo(foo)
  - setBar(bar)
  - lightweight
  - local
  - arbitrary state

- **Message**
  - createBuilder()
  - getMessage()
  - threadsafe
  - valid
Sample Packet Loop

```java
OFConnection conn = switch.getConnection();
while (true) {
    // read from wire
    OFGenericMessage message = conn.readMessage;

    switch (message.getType()) {
    case ECHO_REQUEST:
        send(message.asEchoRequest().createReply());
        break;
    case PACKET_IN:
        handlePacketIn(conn, message.asPacketIn());
        break;
    case FLOW_STATS_REPLY:
        updateStats(message.asFlowStatsReply());
        break;
    }
```

Wire Transfer is done by `messageFactory`

GenericMessage has `getType()` and `as${type}() createReply()` convenience function
private Match createMatch(OFMessageFactory fact, OFPacketIn packetIn) {
    MatchBuilder builder = factory.createMatchBuilder();

    if (builder.supportsField(IN_PORT))
        builder.setField(IN_PORT, OFPort.of(12));

    if (builder.supportsMask(ETH_DST))
        builder.setMasked(ETH_DST, packetIn.getMatch().getMasked(ETH_DST));

    if (builder.supportsWildcards(IPV6_DST))
        builder.setMaskedMatch(IPV6_DST, IPv6Mask.of("00:01:02::/64"));

    IPv4 field = builder.getField(IPV4_SRC);
}

Match builder can be queried for supported fields

ipv4 field = builder.getField(IPV4_SRC);
Create a FlowMod

```java
private void handlePacketIn(OFConnection conn, OFPacketIn packetIn) {

    OFMessageFactory factory = conn.getMessageFactory();
    OFFlowModBuilder fb = factory.createFlowModBuilder();

    fb.setCookie(123).setMatch(createMatch());
    fb.addAction(messageFactory.actionOutput(10));
    OFFlowMod flowMod = fb.getMessage();
    send(flowMod);

    OFFlowMod modifiedFlowMod =
        flowMod.createBuilder().setInport(12).getMessage();
    send(modifiedFlowMod);
}
```

builders are mutable, single threaded + dirt cheap to construct

Builder used inline to 'modify' (create modified copy) of msg
Summary

• OpenFlowJ/Loxi: major update to Floodlight’s OpenFlow API
  • OF multi version support (but doesn’t manage the differences for you)
  • Bug resilience

• 1st Milestone: 2Q

• Check out a demo of the client side API

https://github.com/andi-bigswitch/openflowj-demo
<table>
<thead>
<tr>
<th>Time</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>8:30-9:00a</td>
<td>Open SDN Stack Overview</td>
</tr>
<tr>
<td>9:00-10:00a</td>
<td>Open Source Switch OS: Open Network Linux</td>
</tr>
<tr>
<td>10:00-11:30a</td>
<td>OF-DPA: Open API for Broadcom StrataXGS Architecture</td>
</tr>
<tr>
<td><strong>11:30-12:30p</strong></td>
<td><strong>Lunch</strong></td>
</tr>
<tr>
<td>12:30-1:00p</td>
<td>Indigo OpenFlow 1.3 Architecture</td>
</tr>
<tr>
<td>1:00-2:00p</td>
<td>Floodlight, LOXI, and OpenFlow 1.3 Support</td>
</tr>
</tbody>
</table>
Conclusion: Open SDN Stack

• Phew! – Large Brain Dump
• But, first top-to-bottom Open SDN stack
  • Hardware, OS, ASIC SDK, OpenFlow Agent, OpenFlow Driver, SDN Controller
• Thanks!
• If you’re not bored of me, join me in the Research Track 😊
Thank You