AGENDA / OUTLINE

• What is Open Network Linux? What is it good for?

• ONL source code and build system overview

• Writing to ONL: ONLP and ONLSwitch APIs

• General discussion
  – Pro's and Con's of different hardware platforms
  – Other discussion
ONL: OPEN NETWORK LINUX
A Linux Distribution for Bare Metal Switches

• Reference NOS for the Open Compute Project (OCP)
  – Collection of software packages, utilities, drivers, and abstractions to run on OCP, bare metal, “brite box” hardware
  – i.e., a “NOS” that ONIE would install

• OCP Networking Community/Ecosystem

http://opennetlinux.org
ONL: EXAMPLE USE-CASES

• Reference hardware testing platform, e.g., OCP Certification
  – Common core in open source for verification, reference
  – Shared platform code ensures consistency and test coverage in areas like cable/optics compatibility, operational statistics, front panel LED behavior, SNMP behavior, environmental sensors, fan drivers, etc.

• DIY packet forwarding platform, e.g., for academic research
  – Platform driver code too complicated for most hobbyists
  – Pluggable forwarding agents enable innovation in dataplane features and network designs

• Building Block for Commercial or Production-grade Software
  – Pull ONL in to a larger, commercially supported solution – e.g. Big Switch products
  – Interest in using ONL as the basis for web-scale OS’es – e.g. NTT Labs / Facebook
Big Switch uses ONL as the base for Switch Light™ OS

Alternative to NPB:
- Gigamon
- Netscout
- Anue
- IXIA

Alternative to ACI:
- Cisco
- Apic

1G/10G/40G
10G/40G (Trident-II)
WHY OPEN NETWORK LINUX?

• Why not use an existing Linux distribution?
  – Switches are very similar to servers, but not quite
  – Does build on existing distribution – Debian Wheezy, Jessie
  – Need to create ONIE installers for many platforms
  – Need to manage switch-specific hardware (e.g., SFPs)

• Distribution or ‘A La Carte’?
  – ONL supports both
  – This talk focuses on the ‘A La Carte’ pieces of ONL
## ONL: SUPPORTED HARDWARE

http://opennetlinux.org/hcl

### ONL Supports 25+ Switches
- x86, PowerPC, and ARM Architectures
- 1g, 10g, 40g, and 100g Switches
- Many Open Compute Project switches
- Many non-OCP switches

### Platform Ports in Progress
- Freescale T2080RDB
- Accton AS4610 (cost optimized)
- Facebook Wedge 100
- Celestica Redstone XP
- Accton AS7512 w/ Cavium processor

---

**Quanta**

<table>
<thead>
<tr>
<th>Device</th>
<th>Ports</th>
<th>CPU</th>
<th>Forwarding</th>
<th>Support Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quanta-T048cR-L89</td>
<td>48x1G + 4x10G</td>
<td>Freescale P9200</td>
<td>Broadcom BCM55241 (FireboltII)</td>
<td>Supported and Tested</td>
</tr>
<tr>
<td>Quanta-T048cR-L99a</td>
<td>48x1G + 4x10G</td>
<td>Freescale P9200</td>
<td>Broadcom BCM55241 (FireboltII)</td>
<td>Supported, no ONL, not regularly tested</td>
</tr>
<tr>
<td>Quanta-T048cR-L72</td>
<td>48x1G + 4x10G</td>
<td>Freescale P9200</td>
<td>Broadcom BCM55241 (FireboltII)</td>
<td>Supported and Tested</td>
</tr>
<tr>
<td>Quanta-T048cR-L78</td>
<td>48x1G + 4x10G</td>
<td>Intel Xeon 5218</td>
<td>Broadcom BCM55241 (FireboltII)</td>
<td>Supported and ran in the lab</td>
</tr>
<tr>
<td>Quanta-T032-L79</td>
<td>32x40G</td>
<td>Intel Xeon 5218</td>
<td>Broadcom BCM55241 (FireboltII)</td>
<td>Supported and ran in the lab</td>
</tr>
</tbody>
</table>

**Accton/Edge-Core**

<table>
<thead>
<tr>
<th>Device</th>
<th>Ports</th>
<th>CPU</th>
<th>Forwarding</th>
<th>Support Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accton-AG4000-4X1T</td>
<td>48x1G + 4x10G</td>
<td>Freescale P9200</td>
<td>Broadcom BCM55241 (Agile3)</td>
<td>Supported and Tested</td>
</tr>
<tr>
<td>Accton-AG4000-4X2K</td>
<td>48x1G + 4x40G</td>
<td>Freescale P9200</td>
<td>Broadcom BCM55241 (Triband)</td>
<td>Supported and Tested</td>
</tr>
<tr>
<td>Accton-AG4110-4X2K</td>
<td>48x1G + 4x40G</td>
<td>Freescale P9200</td>
<td>Broadcom BCM55241 (Triband)</td>
<td>Supported and Tested</td>
</tr>
<tr>
<td>Accton-AS710-4X4</td>
<td>48x1G + 4x40G</td>
<td>Freescale P9200</td>
<td>Broadcom BCM55241 (Triband)</td>
<td>Supported and Tested</td>
</tr>
<tr>
<td>Accton-AS700-4X4</td>
<td>48x1G + 4x40G</td>
<td>Freescale P9200</td>
<td>Broadcom BCM55241 (Triband)</td>
<td>Supported and Tested</td>
</tr>
<tr>
<td>Accton-AS712-4X4K</td>
<td>48x1G + 4x10G</td>
<td>Intel Xeon 5218</td>
<td>Broadcom BCM55241 (Triband)</td>
<td>Supported and Tested</td>
</tr>
<tr>
<td>Accton-AS712-4X8K</td>
<td>32x40G</td>
<td>Intel Xeon 5218</td>
<td>Broadcom BCM55241 (Triband)</td>
<td>Supported and Tested</td>
</tr>
</tbody>
</table>

**DNI/Aigma**

<table>
<thead>
<tr>
<th>Device</th>
<th>Ports</th>
<th>CPU</th>
<th>Forwarding</th>
<th>Support Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>AG-T448CU</td>
<td>48x1G + 4x40G</td>
<td>Freescale P9200</td>
<td>Broadcom BCM55241 (Triband)</td>
<td>Supported and Tested</td>
</tr>
</tbody>
</table>

**Dell**

<table>
<thead>
<tr>
<th>Device</th>
<th>Ports</th>
<th>CPU</th>
<th>Forwarding</th>
<th>Support Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>S4810-ON</td>
<td>48x1G + 4x40G</td>
<td>Freescale P9200</td>
<td>Broadcom BCM55241 (Triband)</td>
<td>Supported and Tested</td>
</tr>
<tr>
<td>S4348-ON</td>
<td>48x1G + 4x40G</td>
<td>Intel Xeon E3-1238</td>
<td>Broadcom BCM55241 (Triband)</td>
<td>Supported and Tested</td>
</tr>
<tr>
<td>S6048-ON</td>
<td>32x40G</td>
<td>Intel Xeon 5218</td>
<td>Broadcom BCM55241 (Triband)</td>
<td>Supported and Tested</td>
</tr>
</tbody>
</table>
ONL: THREE+ FORWARDING AGENTS; 20+ PLATFORMS

FORWARDING AGENT PROGRAMMING APIS

Installer (ONIE)

ONL Linux Kernel
(Includes extra drivers: I2C, MUX, management Ethernet, …)

Open Network Linux Platform Abstraction Layer
(Platform specific drivers including Optics)

Platform Driver

PLATFORM

Microsoft SONiC

Facebook FBOSS

Indigo OpenFlow Agent

HARDWARE

CPU
(PowerPC, x86)

Miscellaneous Hardware
(Fans, LED controllers, SFP sensors, Power…)

Packet Forwarding Chip
(ASIC)

OCP Switch Hardware
(Facebook Wedge, IM Niagara, Accton 6712, 7512, Dell S6000-ON, Quanta LY6, …)

Broadcom SDK
(others coming soon)

SAI Interface

Open NSL

OF-DPA

ADDITIONAL / FUTURE APPLICATIONS HERE
• What is Open Network Linux? What is it good for?

• ONL source code and build system overview

• Writing to ONL: ONLP and ONLSwitch APIs

• General discussion
  – Pro's and Con's of different hardware platforms
  – Other discussion
ONL: BUILD SYSTEM UNDER THE HOOD
Anatomy of `make onl-$arch`

Kernel Patches

Architecture Infrastructure Module (AIM)

BigCode Libraries (VPI, uCLI, ELS)

Specific Components: `make -C $ONL/packages/base/$arch/$component deb`

Busybox Buildroot Etc.

Example Agents (SONIC, FBOSS, Indigo)

FaultD

Platform.* (installers, etc.)

ONLP/ONLSwitch (Platform API Lib)

OpenNSL/OFDPA

ONLP-SNMP

ONL Loader

Linux 3.2.65

Linux 3.9.6

ONL-Specific .debs $ONL/REPO/$arch

Generic .debs (Emdebian.org & Debian.org) via apt-cacher-ng

SWI:

`make -C $ONL/builds/$arch/swi`

Switch Image (.swi)

Installer Script + SHAR Wrapper

Installer:

`make -C $ONL/builds/$arch/installer`

Open Network Linux + ONIE Installer

New from ONL

3rd Party OSS

Combination

Example Agents

Switch Image

Installer Script + SHAR Wrapper

ONL Loader

© 2015 BIG SWITCH NETWORKS, INC.
ONL BUILD LAYOUT

• cd $ONL & `make docker`
  – Use a docker image ($ONL/docker/images/*) to fix build environment, e.g., cross compiles, libraries, etc.
  – Checks out $ONL/sm/infra (for AIM) and $ONL/sm/bigcode
  – Builds everything at once

• Interactive building:
  – cd $ONL
  – ./docker/tools/onlbuilder  # add “-8” for jessie, defaults to wheezy
  – . setup.env  # source the $ONL/setup.env file into your shell

• ./RELEASE/$suite/$arch/ONL-*.{installer, swi}
  – Resulting images on a successful build
ONL CODE LAYOUT

- `$ONL/packages/base/$arch`
  - `$arch = ‘any’` → “common” across multiple architectures
  - `$arch = ‘all’` → really no arch dependence (e.g., python code)

- `$ONL/packages/platforms/$vendor/$arch/$ONIE_ID`
  - `packages/platforms/accton/x86-64/x86-64-accton-as7712-32x`
  - `./onlp` – the ONLP sub-driver for this platform
  - `./platform-config/r$version/` – installation, booting, initialization
  - More info in `$ONL/docs/PortingGuide.md`

- `$ONL/packages/platforms-platforms-closed`
  - For switch vendors who have not yet open sourced their code
  - E.g., Dell S4048-ON, Dell S6000-ON
ONL DISTRIBUTION INFORMATION

• Default root fs contents:
  – $ONL/builds/any/$suite/common/*-packages.yml

• $ONL/tools
  – onlpm -- used by build system to build packages, extract deps
  – onlrfs -- used to create root file system for distro from .yml file

• $ONL/make
  – Many useful Makefile functions: kernels, .deb pkgs, etc.
  – e.g., set RFS_CONFIG=your-pkgs.yml, include $ONL/make/rfs.mk

• $ONL/REPO/$suite/packages/$arch → resulting .deb files
  – Great for ‘a la carte’ usage
UNDERSTANDING JTABS

• Much of our code has a structured format – affectionately “JTABS”
  – $package/src/module/{auto,src,inc/$package}

• $package/src/module/auto/$package.yml
  – Auto-generate code (parameters, enumerations, etc.) for this module
  – Inline auto-gen code into files via taglets, e.g., /* <auto.start.VAR(PARM) > */

• $package/src/module/src
  – $package_ucli.c: mixin for uCLI command line options
  – $package_log.c: mixin for AIM logging – per-module log levels, etc.
  – $package_config.c: auto-gen, defaults for compile time parameters

• $package/src/utest
  – unit tests for this module

• $(DEPENDMODULES) + dependmodules.mk
  – Makefile variable to pull lists of modules together into a single package
STRATEGIES FOR ROLLING YOUR OWN DISTRO

1. Add the ONL repo as a submodule to your existing code
2. Create a your-pkgs.yml file with custom package lists
   – Mix and match with existing ONL packages as desired
   – Optionally transitively include existing lists, e.g., $arch-common.yml
3. Call the $ONL/tools/onlRfs.py tool to make the root file system
4. Convert the root file system into an ONIE shar (using ONL tools)
5. Possibly re-use existing Makefile magic from $ONL/make

Result: Custom ONIE installer - your-custom-$version.installer
INSTALL USING ONIE THEN BOOT ONL

Boot Logic:

1. uBoot/grub 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

Mass Storage

~2GB

ONL Loader
ONL SWI #1 (cached)
ONL SWI #2 (cached)
• What is Open Network Linux? What is it good for?

• ONL source code and build system overview

• Writing to ONL: ONLP and ONLSwitch APIs

• General discussion
  – Pro's and Con's of different hardware platforms
  – Other discussion
LIBONLP.SO: PLATFORM ABSTRACTION LAYER

• Manages platform peripherals, FRUs, etc.
  – e.g., SFP’s, Fans, Temp Sensors, Power supplies, system details
  – Not any of the Broadcom chip specifics (that’s ONLSwitch.so)

• Full “application facing” API in
  – $ONL/packages/base/any/onlp/src/onlp/module/inc/ONLP/*.h
  – e.g., int onlp_sfp_eeprom_read(int port, uint8_t** rv);
  – e.g., int onlp_fan_info_get(onlp_oid_t id, onlp_fan_info_t* rv);

• onlp_init():
  – Required before any other onlp calls

• onlp_sys_platform_manage_start(int block)
  – Starts thread to manage fans, FRU insertion, etc.
LIBONLP.SO IMPLEMENTATION

Platform Abstraction Layer

• /lib/$platform/libonlp.so depends on:
  – /lib/$platform/libonlp-platform.so # platform specific driver
  – /lib/$platform/libonlp-defaults.so # default implementations

• libonlp-platform.so is a boot-time symlink to
  – /lib/platform-config/$platform/onl/lib/libonlp-$platform.so

• Writing an ONLP driver for a new platform is easy:
  – Driver-side libonlp API is in onlp/inc/ONLP/platformi/*.h
  – Driver can be user-space or kernel-space
  – For more information, see ./docs/PortingGuide.md
LIBONLS.SO: LOW-LEVEL DATAPLANE ABSTRACTION LAYER

• Runs on top of SDK; Provides add-on APIs for your app
• Handles all of the ASIC/SDK initialization
  – Equivalent to config.bcm and rc.soc for all supported platforms
  – Lane mappings, warp core programming, PHY+power settings, etc.
• Provides a unified Port Lifecycle/Management API
  – Operational and Admin state: “link down” versus “admin down”
  – SFF state management: automatic PHY program on SFF insertion
  – Standardized capabilities, capacities, break-out + 1G support
  – LED programming by port state
LIBONLS.SO: PORT MANAGEMENT ABSTRACTIONS

• All port management through serialized event manager
  – Avoids concurrency issues
  – Decouples forwarding logic from platform-level details

• “Switch” Port versus “Platform” port abstraction
  – 1:1 unless breakout cables
  – “Switch” port: logical switching interface
  – “Platform” port: physical port interface

• Application registers for and posts port events to ONLS
  – Status: Link up/down, SFF insert/removal, breakout, auto-neg, etc.
  – Operational: Admin up/down, disable auto-neg, etc.
LIBONLS.SO: EXAMPLE API SEQUENCE

• Initialize platform:
  – int onls_init();
  – Same for all platforms

• Register for port events:
  – int onls_port_event_register(brcm_port_event_handler_f, void* cookie);

• Post event to ONLS (e.g., ADMIN up a port):
  – ONLS_SWITCH_PORT_EVENT(port, ADMIN_ENABLE);

• Full list of implemented port events in Appendix
AGENDA / OUTLINE

• What is Open Network Linux? What is it good for?

• ONL source code and build system overview

• Writing to ONL: ONLP and ONLSwitch APIs

• General discussion
  – Pro's and Con's of different hardware platforms
  – Other discussion
ONL BRINGS EXPANDED AND UNIFORM PLATFORM SUPPORT

ONL will bring OF-DPA and OpenNSL to 12+ new platforms

<table>
<thead>
<tr>
<th>Switch</th>
<th>OF-DPA Support?</th>
<th>OpenNSL Support?</th>
<th>ONL Support?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accton AS4600, 4610, 5600, 5610, 5710, 5712, 6700, 6712</td>
<td><img src="image1" alt="Support Levels" /></td>
<td><img src="image2" alt="Support Levels" /></td>
<td><img src="image3" alt="Support Levels" /></td>
</tr>
<tr>
<td>Accton Wedge</td>
<td><img src="image4" alt="Support Levels" /></td>
<td><img src="image5" alt="Support Levels" /></td>
<td><img src="image6" alt="Support Levels" /></td>
</tr>
<tr>
<td>Celestica Redstone XP</td>
<td><img src="image7" alt="Support Levels" /></td>
<td><img src="image8" alt="Support Levels" /></td>
<td><img src="image9" alt="Support Levels" /></td>
</tr>
<tr>
<td>IM Niagara</td>
<td><img src="image10" alt="Support Levels" /></td>
<td><img src="image11" alt="Support Levels" /></td>
<td><img src="image12" alt="Support Levels" /></td>
</tr>
<tr>
<td>Quanta LB9, LY2, LY2R, LY6, LY8</td>
<td><img src="image13" alt="Support Levels" /></td>
<td><img src="image14" alt="Support Levels" /></td>
<td><img src="image15" alt="Support Levels" /></td>
</tr>
<tr>
<td>Dell S4048, S4810, S6000, Z9100</td>
<td><img src="image16" alt="Support Levels" /></td>
<td><img src="image17" alt="Support Levels" /></td>
<td><img src="image18" alt="Support Levels" /></td>
</tr>
</tbody>
</table>

ONL also adds support for:

- Break-out cables
- 1G optics
- Dynamic re-cabling (breakout to standard)
- Additional cables, optics

Partial; Full Support in Progress
DISCUSSION TOPICS

• Roll your own SDK or use our’s?
• OpenNSL versus the direct SDK?
• First target project?
• Is our Indigo top-half of value?
• Interaction model?
  – Is the public mailing list enough?
• Testing and support model?
• Tour of the data center downstairs for testing
CONCLUSION AND HOW TO GET INVOLVED

• Join the mailing list and provide feedback
  – Compile the code
  – Try a new platform
  – “Feedback is fine, but patches are preferred” 😊

• Growing support for OCP and non-OCP switches

• Forwarding Agents: Fboss, ORC, OpenFlow and more

• Find out more at http://opennetlinux.org
  • Documentation/Videos
  • Tutorials: http://opennetlinux.org/docs/routingtutorial
  • Pre-compiled binaries
LIBONLS.SO: LIST OF PORT EVENT

- ONLS_PORT_EVENT_PLATFORM_PORT_ACTION_CONFIGURE
- ONLS_PORT_EVENTPLATFORM_PORT_ACTION_ADMIN_ENABLE
- ONLS_PORT_EVENTPLATFORM_PORT_ACTION_ADMIN_DISABLE
- ONLS_PORT_EVENTPLATFORM_PORT_ACTION_SFP_INSERT
- ONLS_PORT_EVENTPLATFORM_PORT_ACTION_SFP_REMOVE
- ONLS_PORT_EVENTPLATFORM_PORT_ACTION_SFP_IDENTIFIED
- ONLS_PORT_EVENTPLATFORM_PORT_ACTION_SFP_SUPPORTED
- ONLS_PORT_EVENTPLATFORM_PORT_ACTION_SFP_UNSUPPORTED
- ONLS_PORT_EVENTPLATFORM_PORT_ACTION_SFP_FORCE_ABSENT_ENABLE
- ONLS_PORT_EVENTPLATFORM_PORT_ACTION_SFP_FORCE_ABSENT_DISABLE
- ONLS_PORT_EVENTPLATFORM_PORT_ACTION_FORCE_ONLINE_ENABLE
- ONLS_PORT_EVENTPLATFORM_PORT_ACTION_FORCE_ONLINE_DISABLE
- ONLS_PORT_EVENTPLATFORM_PORT_ACTION_SFP_ALWAYS_ENABLED_ENABLE
- ONLS_PORT_EVENTPLATFORM_PORT_ACTION_SFP_ALWAYS_ENABLED_DISABLE
- ONLS_SWITCH_PORT_ACTION_ADD
- ONLS_SWITCH_PORT_ACTION_REMOVE
- ONLS_SWITCH_PORT_ACTION_CONFIGURE
- ONLS_SWITCH_PORT_ACTION_ADMIN_ENABLE
- ONLS_SWITCH_PORT_ACTION_ADMIN_DISABLE
- ONLS_SWITCH_PORT_ACTION_PLATFORM_ENABLE
- ONLS_SWITCH_PORT_ACTION_PLATFORM_DISABLE
- ONLS_SWITCH_PORT_ACTION_LINKSCAN
- ONLS_SWITCH_PORT_ACTION_LINK_UP
- ONLS_SWITCH_PORT_ACTION_LINK_DOWN
- ONLS_SWITCH_PORT_ACTION_LINK_REMOTE_FAULT
- ONLS_SWITCH_PORT_ACTION_LINK_FAILED
- ONLS_SWITCH_PORT_ACTION_AUTONEG_ENABLE
- ONLS_SWITCH_PORT_ACTION_AUTONEG_DISABLE
- ONLS_SWITCH_PORT_ACTION_LINK_UP
- ONLS_SWITCH_PORT_ACTION_LINK_DOWN
- ONLS_SWITCH_PORT_ACTION_LINK_REMOTE_FAULT
- ONLS_SWITCH_PORT_ACTION_LINK_FAILED
- ONLS_SWITCH_PORT_ACTION_AUTONEG_ENABLE
- ONLS_SWITCH_PORT_ACTION_AUTONEG_DISABLE
- ONLS_SWITCH_PORT_ACTION_LINK_UP
- ONLS_SWITCH_PORT_ACTION_LINK_DOWN
- ONLS_SWITCH_PORT_ACTION_LINK_REMOTE_FAULT
- ONLS_SWITCH_PORT_ACTION_LINK_FAILED
ONL: EXAMPLE INDUSTRY USE CASES

ON.LAB

Project Atrium with Indigo OpenFlow Agent

Facebook

FBOSS – Built on Broadcom’s OpenNSL Library

NTT

L3 Routing using GoBGP + ORC

Details in following slides.
ONL/OpenFlow on Project Atrium

**ONOS**

**E-BGP**

**Quagga BGP**

Peering Application

ONOS

OFDPA Driver

OpenFlow 1.3

Indigo OF Agent

**OCP Software**

- OF-DPA API
- OF-DPA
- BRCM SDK API
- BRCM ASIC

**OCP Bare Metal Hardware**

**OCP**: Open Compute Project; **ONL**: Open Network Linux; **ONIE**: Open Network Install Env; **BRCM**: Broadcom Merchant Silicon ASICS; **OF-DPA**: OpenFlow Datapath Abstraction

**Entities**

- vlan x
- vlan y
- vlan z
- E-BGP

**Partners**

- Accton
  - Making Partnership Work
- BRCM
- Indigo OF Agent
- ONIE
- ONL
- OPEN Compute Project
- Quanta

**Projects**

- BRCM SDK API
- BRCM ASIC
- OCP Bare Metal Hardware
- OCP Software
- Quagga BGP
- Peering Application
- ONOS
- OFDPA Driver
- OpenFlow 1.3
- Indigo OF Agent
- OF-DPA API
- OF-DPA
- BRCM SDK API
- BRCM ASIC
- OCP Bare Metal Hardware
- OCP Software
FACEBOOK FBOSS SUPPORT ONL IN ALPHA

40G Breakout Cable

Vlan #1

Interfaces

Vlan #2

Hosts

10G SFP+ Cables

Demo From OCP Workshop 10/9 in Boston
NTT’S EVPN USE CASE

- ospfd
- GoBGP
- zapi
- zebra
- ORC
- OpenNSL
- Linux Networking Subsystem
- netlink
- driver
- ONL
- ONIE
- Accton’s Edgecore 5712-54x

Copyright©2015 NTT Corp. All Rights Reserved.
root@accton1:~# ifconfig | less
root@accton1:~# ip route show
default via 10.0.0.1 dev m0l
10.0.0.0/8 dev m0l proto kernel scope link src 10.129.7.12
192.168.0.0/24 dev orc01 proto kernel scope link src 192.168.0.1
192.168.1.0/24 dev orc02 proto kernel scope link src 192.168.1.1

root@accton1:~#
### NTT GOBGP DEMO – GOBGP RUNNING

```
root@accton1:~# gobgp global rib
Network not in table
```

```
root@accton1:~# gobgp neighbor
Peer   AS  Up/Down State    #Advertised  Received  Accepted
192.168.0.2  65002 00:01:39 Establ 0 0 0
```

```
root@accton2:~# gobgp neighbor
Peer   AS  Up/Down State    #Advertised  Received  Accepted
192.168.0.1  65001 00:01:45 Establ 0 0 0
```

```
soramichi@server1:~$
```
root@accton1:~# gobgp global rib add 192.168.1.0/24
root@accton1:~# ip route show
default via 10.0.0.1 dev m0
10.0.0.0/8 dev m0 proto kernel scope link src 10.129.7.12
192.168.0.0/24 dev orc01 proto kernel scope link src 192.168.0.1
192.168.1.0/24 dev orc02 proto kernel scope link src 192.168.1.1
192.168.2.0/24 via 192.168.0.2 dev orc01 proto zebra

root@accton1:~#

root@accton2:~# gobgp global rib add 192.168.2.0/24
root@accton2:~#
```bash
root@accton1:~# gbgp global rib add 192.168.1.0/24
root@accton1:~# ip route show
default via 10.0.0.1 dev m1
t 0.0.0.0/8 dev m1 proto kernel scope link src 10.129.7.12
192.168.0.0/24 dev orc01 proto kernel scope link src 192.168.0.1
192.168.1.0/24 dev orc02 proto kernel scope link src 192.168.1.1
192.168.2.0/24 dev orc01 proto kernel scope link src 192.168.2.1

root@accton2:~# gbgp global rib add 192.168.2.0/24
root@accton2:~# ip route show
default via 10.0.0.1 dev m1
t 0.0.0.0/8 dev m1 proto kernel scope link src 10.129.7.13
192.168.0.0/24 dev orc01 proto kernel scope link src 192.168.0.2
192.168.1.0/24 via 192.168.0.1 dev orc01 proto zebra
192.168.2.0/24 dev orc02 proto kernel scope link src 192.168.2.1

root@accton1:~# gbgp global rib add 192.168.1.0/24
```

```
grep 19 BGP, length: 19
07:02:37:137996 IP 192.168.0.2.41697 > 192.168.0.1.bgp Flags [P.], seq 86, ack 85, win 115, options [nop,nop,TS val 4768952 ecr 47237742], length 19 BGP, length: 19
07:02:37:136835 IP 192.168.0.1.bgp > 192.168.0.2.41697: Flags [., ack 104, win 115, options [nop,nop,TS val 47240925 ecr 47068952], length 0
07:02:37:138106 IP 192.168.0.2.41697 > 192.168.0.1.bgp: Flags [., ack 104, win 115, options [nop,nop,TS val 47068952 ecr 47240925], length 0
07:02:42:141295 ARP, Request who-has 192.168.0.2 tell 192.168.0.1, length 28
07:02:42:141800 ARP, Reply 192.168.0.2 is-at 2a:83:66:c1:f0:15 (oui Unknown), length 50
```

```
soramichi@server1:~$ iperf -c 192.168.2.2
Client connecting to 192.168.2.2, TCP port 5001
TCP window size: 85.0 KByte (default)
[ 3] local 192.168.1.2 port 46110 connected with 192.168.2.2 port 5001

soramichi@server2:~$ iperf -s
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
[ 4] local 192.168.2.2 port 5001 connected with 192.168.1.2 port 46110
```
NTT GOBGP demo – 9.41 Gbits/sec

root@acton1:~# gbgp rib add 192.168.1.0/24
root@acton1:~# ip route show
default via 10.0.0.1 dev m0
10.0.0.0/8 dev m0 proto kernel scope link src 10.129.7.12
192.168.0.0/24 dev orco1 proto kernel scope link src 192.168.0.1
192.168.1.0/24 dev orco2 proto kernel scope link src 192.168.1.1
192.168.2.0/24 via 192.168.0.2 dev orco1 proto zebra

root@acton1:~#

root@acton2:~# gbgp rib add 192.168.2.0/24
root@acton2:~# ip route show
default via 10.0.0.1 dev m0
10.0.0.0/8 dev m0 proto kernel scope link src 10.129.7.13
192.168.0.0/24 dev orco1 proto kernel scope link src 192.168.0.2
192.168.1.0/24 via 192.168.1.0 dev orco1 proto zebra
192.168.2.0/24 dev orco2 proto kernel scope link src 192.168.2.1

root@acton2:~#

length 19: BGP, length: 19
07:02:37.137996 IP 192.168.0.2.41697 > 192.168.0.1.bgp: Flags [P.], seq 86
105, ack 85, win 115, options [nop,nop,TS val 4768952 ecr 47237742], len
th 19: BGP, length: 19
07:02:37.138035 IP 192.168.0.1.bgp > 192.168.0.2.41697: Flags [., ack 85
114, options [nop,nop,TS val 47240925 ecr 47689592], length 0
28
07:02:37.138196 IP 192.168.0.2.41697 > 192.168.0.1.bgp: Flags [., ack 104
114, options [nop,nop,TS val 47689592 ecr 47240925], length 0
28
07:02:42.141295 ARP, Request who-has 192.168.0.2 tell 192.168.0.1, length
28
07:02:42.141800 ARP, Reply 192.168.0.2 is-at 2a:83:66:c1:f0:15 (oui Unknown)
length 50

soramichi@server1:~$ iperf -c 192.168.2.2
Client connecting to 192.168.2.2, TCP port 5001
TCP window size: 85.0 KByte (default)

server listening on TCP port 5001
TCP window size: 85.3 KByte (default)

[ 3] local 192.168.1.2 port 46110 connected with 192.168.2.2 port 5001
[ ID] Interval Transfer Bandwidth
[ 3] 0.0-10.0 sec 11.0 GBytes 9.42 Gbits/sec

soramichi@server1:~$
Early Adoption: REANNZ

• Research and Education Advanced Network New Zealand
  – R&E WAN Provider for NZ’s University Systems
• REANNZ is currently trialing OF-DPA on Open Network Linux
  – Previously trialed many SDN platforms
  – CARDIGAN, Front Line Assembly and Treehouse
• “A Linux for Networking”
• REANNZ is working on new SDN deployment options
ONL Contributors/Supporters (So Far)

Adds Forwarding Agents

Open Network Linux

Provides Platform Drivers

© 2015 BIG SWITCH NETWORKS, INC.
Early Adoption: REANNZ

• Research and Education Advanced Network New Zealand
  – R&E WAN Provider for NZ’s University Systems
• REANNZ is currently trialing OF-DPA on Open Network Linux
  – Previously trialed many SDN platforms
  – CARDIGAN, Front Line Assembly and Treehouse
• “A Linux for Networking”
• REANNZ is working on new SDN deployment options
OPEN NETWORK INSTALL ENVIRONMENT (ONIE)

• Open source project to install/uninstall network OS
  • http://github.com/onie/onie or http://onie.github.io/onie/

• 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
Traditional: ONL + Netlink w/Open Route Cache

This API looks a lot like SAI

Applications

- SensorD
- ONLP Dump
- SNMP
- Open Route Cache
- Orc-brcm Driver

Platform

- ONLP APIs
- ONL Platform APIs
- ONL Distribution (Linux + stuff)
- BRCM SDK

Hardware

- CPU (x86, PPC)
- Misc Hardware (fans, LEDs, SFP, sensors)
- BRCM

Closed Source

Per Switch Stack
Basic L3 Routing With Open Route Cache (ORC)

1. ORC creates a orcXX interface for each port

2. Quagga (or Xorp, etc.) installs routes into Linux kernel

3. Linux kernel sends RT Netlink Updates to ORC

4. ORC translates to SDK calls
   - Hardware accelerated routing!
   - Only unicast IPv4 now
   - No IPv6, Multicast, ECMP support

5. Linux kernel is slow path to ASIC’s fast path
Lots of open source controllers

• But what are they controlling?
ONL w/ OpenFlow Agent

OpenFlow Protocol: v1.0-v1.3

Applications
- SensorD
- ONLP Dump
- SNMP
- Indigo OpenFlow Agent
- Indigo OF-DPA Driver

Platform
- ONL Platform APIs
- ONL Distribution (Linux + stuff)
- OF-DPA Driver
- BRCM SDK

Hardware
- CPU (x86, PPC)
- Misc Hardware (fans, LEDs, SFP, sensors)
- BRCM

Per Switch Stack
OpenFlow Datapath Abstraction: OF-DPA

Figure 2: OpenFlow OF-DPA Abstract Switch Pipeline
SWITCH SCALING: UNDERSTANDING TABLES
Classical OpenFlow versus Modern: Proactive and Multi-table OpenFlow

Processing Pipeline

Early OpenFlow implementations only used the most flexible table.
- Didn’t scale well
- Scale forced reactive population
- Reactive caused controller load and addition scale problems

Modern OpenFlow implementation leverages all tables
- Scales like traditional
- Allows proactive population
- Proactive reduces controller load and allows for headless control

OpenFlow does **NOT** imply Flow-based Networking
DISCUSSION: SAI VS. OPENNSL VS. OFDPA

• Short story: this is all evolving
• Closed versus Open Source
• SAI gets you chip vendor independence (in theory)
  – Broadcom, Mellanox, Cadvium, Barefoot (via software)
• OpenNSL is closest to existing BCM APIs
• OF-DPA exposes other functionality like OAM and MPLS
• There is a move to merge these APIs
  – e.g., SAI implemented on OpenNSL, OF-DPA implemented on OpenNSL
Thank You

HTTP://OPENNETLINUX.ORG