A Brief Introduction to OpenDaylight

Colin Dixon
Technical Steering Committee Chair, OpenDaylight
Senior Principal Engineer, Brocade

Some content from: David Meyer, Neela Jaques, and Kevin Woods
Outline

• Introduction...
  • ...to SDN
  • ...to OpenDaylight...
    • ...with a brief aside into YANG

• New in Lithium

• Plans for Beryllium
Traditional SDN (OpenFlow)
The separation of the control and data planes

• Modern switches
  • Control/data plane both on switch
  • Data plane: fast, reads tables
  • Control plane: slow, writes tables

• SDN
  • Decouple control/data planes
  • Data plane on the switch
  • Control plane elsewhere, e.g., an x86 server, can do fancier things
Modern, Inclusive SDN

Vendor A
Vendor B
Vendor C

Logically
Centralized
SDN Controller

Northbound API

Industry Standard
Control/Management
Protocols

Standard
Modeling
Language

Vendor A
Vendor B
Vendor C
What is OpenDaylight

OpenDaylight is an Open Source Software project under the Linux Foundation with the goal of furthering the adoption and innovation of Software Defined Networking (SDN) through the creation of a common industry supported platform.

**Code**

To create a robust, extensible, open source code base that covers the major common components required to build an SDN solution

**Acceptance**

To get broad industry acceptance amongst vendors and users:
- Using it directly or through vendor products
- Vendors using OpenDaylight in commercial products

**Community**

To have a thriving and growing technical community contributing to the code base, using the code in commercial products, and adding value above, below and around.
OpenDaylight Releases

• **Hydrogen** (first release)
  • February 2014
  • 13 projects, 1.3m lines of code

• **Helium** (second release)
  • October 2014
  • 25 projects, 2.1m lines of code

• **Lithium** (latest release)
  • June 2015
  • 40+ projects, 2.3m lines of code
Core Architecture

Model-Driven Service Abstraction Layer (MD-SAL)

- App/Service
- App/Service
- Plugin
- Plugin

Data

- Notifications
- RPCs

Controllers in a Cluster

YANG Models
What is YANG?

• Data modeling language for NETCONF
  • RFC 6020

• Great, what is NETCONF?
  • Think of it as an SNMP replacement with nice features
  • YANG models ~= SNMP MIBs

• OK, fine, but what is YANG?
What is YANG?

• Three core abstractions
  • Data
  • RPCs (just data in and data out)
  • Notifications (just data out)

• So, it’s really all about the data
What does YANG data look like

- **container** ~ struct
- **list** ~ map/dictionary
- **leaf** ~ primitive types
- **grouping** ~ interface

- Others: typedef, pointers, constraints, etc.

```yang
container network-topology {
  list topology {
    key "topology-id";
    leaf topology-id {
      type topology-id;
    }
  }
  list node {
    key "node-id";
    uses node-attributes;
  }
  list link {
    key "link-id";
    uses link-attributes;
  }
}
```
OpenDaylight Community

• Like any Open Source Project, OpenDaylight primarily consists of those who show up to do the work.

• Running around 250 commits per week over 12 months, trending up
  • 30 Days: ~625 commits, ~100 contributors (7/13/15–8/12/15; during a release)
    • Spikes to ~2x this near releases
  • 12 Months: ~13,250 commits, ~365 contributors (8/12/14–8/12/15)

• Strong integration and testing community
  • This stuff really matters

Source: https://www.openhub.net/p/opendaylight
OpenDaylight Community
What’s in OpenDaylight
Kernel

• AAA
• YANG Tools
• OpenDaylight Controller
• MD-SAL
• NETCONF
• ODL Root Parent
Plugins

- BGP
- CAPWAP
- FaaS
- IoTDM
- LACP
- LISP
- OVSDB

- OpenFlow
  - Circuit switching extensions
  - OF-CONFIG
  - Table Type Patterns
- PCEP
- PacketCablePCMM
- SNMP

- SXP
- Secure Network Bootstrapping
- TCPMD5
- USC
- YANG PUBSUB
Services

- Armoury
- Centinel
- Controller Shield
- DIDM
- Messaging4Transport
- NeutronNorthbound
- NeXt

- ODL-SDNi App
- VPNService
- TSDR
- Topology Processing Framework
- Persistence
Applications

- ALTO
- Defense4All
- Group Based Policy (GBP)
- L2 Switch
- NEMO
- NetIDE
- Network Intent Composition
- OpenDaylight dlux
- OpenDaylight Virtual Tenant Network (VTN)
- Reservation
- Service Function Chaining
- SNMP4SDN
- Unimgr
Metaprojects

• Controller Core Functionality Tutorials
• Documentation
• Integration/distribution
• Integration/Packaging
• Integration/Test
• RelEng/Autorelease
• RelEng/Builder
What do people use it for?
OpenDaylight with OpenStack

• Single OpenStack Neutron service proxy
  • Handles most of the bookkeeping

• Choose your implementation
  • Group-based Policy
  • LISP
  • OVSDB
  • VPN Service (only for VPNaaS)
  • VTN

• Check it out (see the links for instructions)

Network-wide Security Policy

• Historically, policy is mostly
  • Rigidly enforced by the physical topology, e.g., firewall at the gateway
  • Configured “dynamically” via box-by-box Access Control Lists (ACLs)

• New policy efforts are changing this
  • Network Function Virtualization (NFV) and Service Function Chaining (SFC)
  • Automatically generated ACLs based on network-wide policy

• OpenDaylight is a proving ground for at least 3 policy-oriented projects
  • Service Function Chaining, Group-Based Policy, and Network Intent Composition
Programmable EMS and/or NMS

• Huge number of southbound protocol drivers
  • OpenFlow, NETCONF, OVSDB, SNMP, BGP, PCEP, PCMM/COPS, etc.

• With a little bit of effort, you can write “shell scripts” for your network to either gather information or automate tasks

• Automate triggering activities based on network events, e.g., quarantine a host with L2 ACLs based on information from an IDS
How to get involved
Ways to get involved

- **IRC:** #.opendaylight on freenode: [http://webchat.freenode.net/](http://webchat.freenode.net/)
- **Mailing lists:** [http://lists.opendaylight.org/](http://lists.opendaylight.org/)
- **Wiki:** [http://wiki.opendaylight.org/](http://wiki.opendaylight.org/)
- **Documentation:** [https://www.opendaylight.org/downloads](https://www.opendaylight.org/downloads)
  - On github: [https://github.com/opendaylight/docs/](https://github.com/opendaylight/docs/)
- **Git/Gerrit:** [http://git.opendaylight.org/](http://git.opendaylight.org/)
  - Create an account: [https://identity.opendaylight.org/carbon/user-registration/user-registration.jsp](https://identity.opendaylight.org/carbon/user-registration/user-registration.jsp)
- **Projects:** [https://wiki.opendaylight.org/view/Project_list](https://wiki.opendaylight.org/view/Project_list)
  - Individual pages have links to meeting times, code, bugs, IRC channels, etc.
- **Meetings:** [https://wiki.opendaylight.org/view/Meetings](https://wiki.opendaylight.org/view/Meetings)
Backup Slides
How to get there from here

• How do we deploy SDN when it’s not green field
  • Because pretty much nothing is actually green field

  • Hybrid switches, hybrid networks, legacy protocols for interoperability, etc.
    • OpenDaylight supports SNMP, BGP, LISP, NETCONF, etc.

• Trust and stability
  • Current networks build on 40 years of code/experience
  • How can SDN compete with that?
    • Borrow good code/ideas from legacy code
    • Provide better visibility, debugging, etc.
    • Model checking, verification, etc.
Centralized vs. Distributed
(Consistency, Clustering and Federation)

• SDN promises a (logically) centralized control plane

• In practice, we have a distributed cluster of controllers, rather than just one so that
  • we can tolerate faults
  • we can scale out our performance
  • in network partitions there are controllers on both sides

• Providing consistency, federation, scale-out, dealing with CAP trade-offs, etc. is **HARD**

[https://www.youtube.com/watch?v=XQ-lnB3x30g](https://www.youtube.com/watch?v=XQ-lnB3x30g)
Hardware Diversity

• OpenFlow 1.0 provided a lowest common denominator API
  • Real hardware is much more diverse
  • and has many more capabilities

• Exposing this diversity without burdening developers with per-device programming is hard

• Some Attempts
  • Programming Protocol-Independent Packet Processors
  • TTPs from the ONF’s FAWG

https://www.youtube.com/watch?v=bca856w_k_o
http://events.linuxfoundation.org/sites/events/files.slides/TTPs%20and%20NBIs%20for%20ods2014-final_0.pdf
Application Composition

• How can we let multiple SDN apps share the network?
  • PC OSes partition and allocate resources
  • You can’t easily partition the network
    • It’s value comes from the fact that it spans everything
    • You can in some cases, e.g., by address space (FlowVisor)

• Some ideas
  • Most apps should be middleboxes, i.e., NFV
    • Simply chain them together in the right order
    • There’s more to it than this, but linear chaining is powerful
  • Other apps are concerned only with the physical path
    • There is hope that conflicts here can be sanely managed