What You Make Possible
Software Defined Networks and OpenFlow
BRKRST-2051
Frank Brockners
Abstract

- Software Defined Networking (SDN) is a new approach to networking, complementing traditional network architectures. SDN aims at the normalization of network configuration and control through open programmatic interfaces to individual network devices as well as to the whole network. SDN incorporates concepts for network and network topology virtualization, and enables customized control planes. The latter allows close alignment of the network forwarding logic to the requirements of applications. OpenFlow is a specification being developed by the Open Networking Foundation (ONF) that defines a flow-based forwarding infrastructure and a standardized application programmatic interface (API) that allows a controller to direct the functions of a switch through a secure channel. This session supplies an overview of the different concepts present in SDN, discusses contributing technologies, and reviews OpenFlow as a protocol. The SDN concept is put into perspective with existing and evolving network architectures and principles.
“…In the SDN architecture, the control and data planes are decoupled, network intelligence and state are logically centralized, and the underlying network infrastructure is abstracted from the applications…”


“…open standard that enables researchers to run experimental protocols in campus networks. Provides standard hook for researchers to run experiments, without exposing internal working of vendor devices……”

http://www.openflow.org/wp/learnmore/
“A way to optimize link utilization in my network, through new multi-path algorithms”

“A way to reduce the CAPEX of my network and leverage commodity switches”

“An open solution for VM mobility in the Data-Center”

“A way to avoid lock-in to a single networking vendor”

“A platform for developing new control planes”

“A solution to build a very large scale layer-2 network”

“A means to do traffic engineering without MPLS”

“A means to scale my fixed/mobile gateways and optimize their placement”

“A way to define virtual networks with specific topologies for my multi-tenant Data-Center”

“A solution to build virtual topologies with optimum multicast forwarding behavior”

“A way to configure my entire network as a whole rather than individual devices”

“A solution to get a global view of the network – topology and state”

“A way to distribute policy/intent, e.g. for DDoS prevention, in the network”

“A way to build my own security/encryption solution, avoiding RSA”

“A way to scale my firewalls and load balancers”

“Develop solutions software speeds: I don’t want to work with my network vendor or go through lengthy standardization.”

“Common Concepts”

“Different Execution Paths”

BRKRST-2051
© 2012 Cisco and/or its affiliates. All rights reserved.
Cisco Public
Agenda
Towards Open Network Environments for SDN

Approaching Open Network Environments for SDN

Open Network Environments Qualities
- APIs
- Agents and Controllers, The OpenFlow Protocol
- Network Infrastructure Virtualization

Open Network Environments and Software Defined Networks
Approaching Open Network Environments
## Business Objectives and Use-Cases

**Simplified Operations**

<table>
<thead>
<tr>
<th>Enhanced Agility</th>
<th>New Business Opportunities</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Automated large-scale Virtualization</strong> (mixed virtual- and physical topologies)</td>
</tr>
<tr>
<td></td>
<td><strong>Seamless Operational Approach to Servers, Storage, Network</strong> (manage network similar to VM/servers)</td>
</tr>
<tr>
<td></td>
<td><strong>Optimal Service Scaling and Placement</strong> (for physical and virtual appliances)</td>
</tr>
<tr>
<td></td>
<td><strong>Seamless Network Policies</strong> (Security, BYOD, DDoS threat mitigation, Green-Networking,..)</td>
</tr>
<tr>
<td></td>
<td><strong>Unified Access – Wireless &amp; Wireline Integration</strong></td>
</tr>
<tr>
<td><strong>Custom monitoring and control applications</strong></td>
<td><strong>Networks as provisioned Services:</strong> Public and Private Cloud</td>
</tr>
<tr>
<td><strong>Introduce new Network Services “at software speeds”</strong></td>
<td><strong>Monetize Over-the-Top Traffic</strong> (traffic profiling,..)</td>
</tr>
<tr>
<td><strong>Customer-specific Algorithms</strong></td>
<td><strong>New Networking Domains:</strong> Machine-to-Machine Networking</td>
</tr>
<tr>
<td><strong>Bump in the Wire:</strong> Custom firewalls and load balancing</td>
<td></td>
</tr>
</tbody>
</table>

**Can the network be more open, programmable and application aware?**

**Without compromising on security, resiliency and maturity?**
Towards Programmatic Interfaces to the Network
Approaching Today’s Application Developer Dilemma

- Many Network Applications today:
  - OTT – for speed and agility
  - Avoid network interaction – complex and slow innovation

- New Model for Network Applications
  - Keep speed and agility
  - Full-duplex interaction with the network across multiple planes – extract, control, leverage network state

A New Programming Paradigm is Needed
Re-assessing the Network Control Architecture

Evolving Design Constraints on the Control Plane

- **Classic generic networks**
  - Operate without communication guarantees
    A distributed system with arbitrary failures, nearly unbounded latency, and highly variable resources on each node in the system
  - Compute the configuration/forwarding-state of each physical device and keep the information up to date as conditions change
    Change of conditions typically detected by the network elements themselves
  - Operate within given network-level protocol (IP, Ethernet, …)

- **Domain specific networks (e.g. Data-Center, SP-Access/Agg,..)**
  - Specific qualities of these networks relax or evolve network design constraints:
    Examples: Well defined topologies; little variety in network device-types; no arbitrary changes in connected end-hosts (change always an outcome of provisioning action),…
  - Independence of network-level protocol (combined L2, L3 service delivery,…)
Cisco’s Approach: The Power of “And”

Open Network Environment

Industry’s Most Comprehensive and Consistent Portfolio

- Hardware + Software
- Physical + Virtual
- Network + Compute

Applications

OPEN NETWORK ENVIRONMENT

- APIs for Cisco’s Infrastructure and OSs
- Network-wide state control: Agents and Controllers
- Network Infrastructure Virtualization

© 2012 Cisco and/or its affiliates. All rights reserved.
Towards the Open Network Environment for SDN

Implementation Perspective: Evolve the Control-Plane Architecture

- Enable modularization and componentization of network control- and data-plane functions, with associated open interfaces. This allows for optimized placement of these components (network devices, dedicated servers, application servers) and close interlock between applications and network functions.

- Anticipated benefits include: Closely align the control plane with the needs of applications, enable componentization with associated APIs, improve performance and robustness, enhance manageability, operations and consistency.
Open Network Environment

Approaching a Definition

- Open Network Environment – Complementing the Intelligent Network
  - *Preserve what is working*: Resiliency, Scale and Security, Comprehensive feature-set
  - *Evolve for Emerging Requirements*: Operational Simplicity, Programmability, Application-awareness

- The Open Network Environment integrates with existing infrastructure
  - Software Defined Network concepts are a component of the Open Network Environment
  - The OpenFlow protocol can be used to link agents and controllers, and as such is component of SDN as well
Programmatic Network Access – Multiple Layers

Full-Duplex access to the network at multiple layers and networking planes

- Enable a holistic Network Programming model
- Leverage and extend infrastructure at pace of the business
- Deploy common applications across all devices
- Extend/upgrade/add features without upgrading the network operating system
- Reduced time to market by leveraging common platform for building services

- Application/Development: Programmatic network automation, e.g. Cisco Pulse, ...
- Management: Automated, policy directed service and cloud management, e.g. NetworkService Manager, OpenStack, ...
- Orchestration: Network wide service access: Optimized paths (PCE), Topology & service selection (NPS/ALTO), MediaTrace, Address mapping, ...
- Network Service: Common control abstractions: Security, Policy, Routing, ...
- Control: Common forwarding abstractions: Data-Path access, Flow-Forwarding, Tunneling, ...
- Forwarding: Device configuration, state monitoring, logging, debugging
- Transport/Device: Application development frameworks, e.g. Spring, ...
Open Network Environment Qualities
Programmatic APIs

- Programmatic APIs
- Agents and Controllers
- Network Infrastructure Virtualization
The Need for Abstractions
Abstractions in Networking

- Data-plane Abstractions – ISO/OSI Layering
  Examples
  - Local best effort delivery (e.g., Ethernet)
  - Global best effort delivery (e.g., IP)
  - Reliable byte-stream (e.g., TCP)
  Data plane abstractions are key to Internet’s success

- Abstractions for the other planes (control, services, management, orchestration,..)
  … are missing
  Consequences include:
  - Notorious difficulty of e.g. network management solutions
  - Difficulty of evolving software for these planes

“Modularity based on abstraction is the way things get done”

Barbara Liskov
Turing Award Winner
“In computer science, **abstraction** is the process by which data and programs are defined with a representation similar in form to its meaning (semantics), while hiding away the implementation details. Abstraction tries to reduce and factor out details so that the programmer can focus on a few concepts at a time. A system can have several abstraction layers whereby different meanings and amounts of detail are exposed to the programmer. For example, low-level abstraction layers expose details of the computer hardware where the program is run, while high-level layers deal with the business logic of the program.”

http://en.wikipedia.org/wiki/Abstraction_Computer_Science
### Approaching abstractions for Networking

- **Abstractions allow the definition of associated APIs**
  - Enable API platform kit across all platforms, to integrate with development environments
  - Accelerate development of network applications: Completely integrated stack from device to network
  - Multiple deployment modes (local and remote (blade/server) based APIs)
  - Multiple Language Support (C, Java, Python…)
  - Integrate with customer development to deliver enhanced routing, forwarding..
APIs make Abstractions available to Programmers

Example: Cisco’s onePK (one Programming Kit)

- Cisco Network Operating System (IOS, IOS-XR, NX-OS)
- Marshall & Transport
- Application
  - C: Presentation Interface and APIs
  - Java: Presentation Interface and APIs
  - ...*: Presentation Interface and APIs
- Presentation Layer with Multiple Language Bindings
  - Internal Network Abstraction
  - Service Implementation
  - Cisco Operating System target

*Note: Initially “C” and “Java” versions of the Presentation API will be provided; Support for additional programming languages is under consideration.
## Example Network Abstractions

API “Service Sets” delivered through Cisco’s onePK (one Platform Kit)

### Base Service Sets

#### Element
- Element Capabilities
- Configuration Management
- Interface/Ports Events
- Location Information

#### Utilities
- Syslog Events and Queries
- AAA Interface
- Path Trace

#### Discovery
- Network Element Discovery
- Service Discovery
- Topology Discovery

#### Developer
- Debug Capabilities
- Tracing Interfaces
- Management Extensions

#### Data Path
- Packet/Flow Classifiers
- Copy/Punt/Inject
- Statistics

#### Policy
- Interface Policy
- Interface Feature Policy
- Forwarding Policy
- Flow Action Policy

#### Routing
- Read RIB Routes
- Add/Delete Application Routes
- RIB Events (Route up/down)
Network APIs and different hosting models

- Process Hosting
- Blade Hosting
- End-Point Hosting

Communications Channel

onePK Abstraction Interface

<table>
<thead>
<tr>
<th>Element</th>
<th>Utilities</th>
<th>Developer</th>
<th>Discovery</th>
<th>Policy</th>
<th>Routing</th>
<th>Datapath</th>
<th>...</th>
</tr>
</thead>
</table>

Cisco Network Operating System (IOS, IOS-XR, NX-OS)
Programmatic Interfaces for Network Automation

Examples

Network Automation

On-Box Scripting
- TCL
- PERL
- Python

XML Mgmt. Interface
- NETCONF & YANG models
- XMPP, REST

Device and Network APIs
- SDK/APIs in multiple languages.
- Extensibility & Manageability

Network Admin
- Quick automation
- Manual or home grown scripts

NM App Developer
- NM app/OSS
- Standards driven
- Typical net mgmt.

App Developer
- VARs/Partners
- Prog. APIs
- Extend and manage

Consistent user experience (models & APIs) across operating systems and platforms
Not all Networking APIs are created the same

Classes of Networking APIs following their Scope

- Classify Networking APIs based on their scope
  - API Scopes: Location independent; Area; Particular place; Specific device
  - Alternate approaches like device/network/service APIs difficult to associate with use cases
  - Location where an API is hosted can differ from the scope of the API

- Different network planes could implement different flavors of APIs, based on associated abstractions

Utility
  - Example: Get Auth, Publish Log...
  - Scope: Location independent

Area/Set
  - Example: Domain, OSPF-area...
  - Scope: Group/Set/Area

Place in the Network
  - Example: Edge Session, NAT
  - Scope: Specific place/location

Element
  - Example: interface statistics
  - Scope: Specific element
APIs at work – Element APIs
Example: Statistics, Diagnostics & Troubleshooting

Objective:
- Provide operators/administrators/support engineers with details about how packets flow through the network.
- Reveal network issues

Approach
- NMS application leverages onePK APIs to show path of flow, timestamp, ingress/egress interfaces, interface packet counts

Example: Statistics, Diagnostics & Troubleshooting

Ingress time: May 15, 2011 00:46:55.145
Ingress intf: Gi0/1
Ingress pkts: 30
Egress time: Jun 6, 2011 00:46:55.251
Egress intf: Gi0/0
Egress pkts: 5
APIs at work – Place in the Network APIs

Example: Dynamic Bandwidth/QoS Allocation

- **Business Problem**
  - Enable superior experience for subscribers which access a particular cloud service

- **Solution**
  - Install customer policy (QoS, access control,..) using **onePK** on key networking elements, e.g. Provider Edge (PE) routers
  - Similarities to broadband “Bandwidth on Demand” use cases
    - Broadband: Policy controlled on Subscriber-Gateway (BRAS/BNG, GGSN/PGW, ..) only
    - Common API like onePK enables control points on all key networking devices
APIs at work – Area APIs

Example: Custom Routing

- **Business Problem**
  - Network operator needs to direct traffic using unique or external decision criteria; e.g. route long lived elephant flows, like backup traffic differently.

- **Solution**
  - Custom route application built and deployed using onePK, communicating directly with the forwarding plane.
  - Unique data forwarding algorithm highly optimized for the network operator’s application.

![Diagram](image)
APIs at work – Area APIs

Examples: Topology graph

- **Business Problem**
  - Several problems require a view of the network topology (area, domain, or whole network)
  - Examples:
    - Locate optimal service out of a given list
    - Optimize Load Placement
    - Visualize the active Network Topology

- **Solution**
  - Topology API to expose network topology to applications, such as
    - NPS (for service selection)
    - Hadoop (for optimal job placement)
    - NMS (for topology visualization)
Open Network Environment Qualities

Agents and Controllers:
*Logically centralized and fully distributed Control*

- Programmatic APIs
- Agents and Controllers
- Network Infrastructure Virtualization
Approaching Agents and Controllers
Agent-Controller pairs: A few well-known and evolving examples

- Agents and Controllers are a component of Open Network Environments and a key component of the evolving “Software Defined Network” concept
- The Concept of Agents and Controllers exists in the Industry for quite some time
- Observation: Current Agent-Controller pairs always serve a specific task (or set of tasks) in a specific domain
Agent and Controllers
Logically centralized control across multiple network elements

- Some network delivered functionality benefits from logically centralized coordination across multiple network devices
  - Functionality typically domain, task, or customer specific
  - Typically multiple Controller-Agent pairs are combined for a network solution

- Controller
  - Process on a device, interacting with a set of devices using a set of APIs or protocols
  - Offer a control interface/API

- Agent
  - Process or library on a device, leverages device APIs to deliver a task/domain specific function
Agent and Controllers

- **Controller** – typical tasks
  - Gather, Analyze, Receive Requests
  - Makes a decision and pushes it to the agent(s); ephemeral and configuration data
- **Agents** – typical tasks
  - Act, Observe, Notify
  - Deployment dependent, Controller can delegate rules to Agent to enable the Agent to take local decisions
Distributed Control
Exploring the tradeoff between Agents and Controllers – and fully distributed Control

- Control loop requirements differ per function/service and deployment domain
  - “As loose as possible, as tight as needed”
  - Latency, Scalability, Robustness, Consistency, Availability
  - Different requirements per use case
    Example:
    Topology for Visualization (Network Management) vs. Topology for Path-Computation/Routing

- How to decide which functionality is well suited a particular control paradigm?

Note: Example only – Not all network planes shown
“Subsidiarity is an organizing principle that matters ought to be handled by the smallest, lowest or least centralized competent authority.”

http://en.wikipedia.org/wiki/Subsidiarity
Decision making feedback loop
Considerations – Applying Subsidiarity to Networking

- Locations of event/state-source, state-processing, decision-making, action-taking can follow specific requirements
  - Fully distributed, agent/controller architectures, etc.

- Different design goals and pre-requisites
  - Required reaction time-scales
    - Fast convergence (e.g. for voice/video apps) vs. conservative reaction times (long running batch-type applications)
  - Leveraging state/information from multiple sources (network and applications) for decision making
  - Macro vs. micro-level decision making (e.g. link/device layer redundancy vs. cluster/POD level redundancy in MSDC)
## Evolving the Control Plane Environment

### Deployment Considerations – Applying Subsidiarity to Networking

| Rapid prototyping (TTM vs. performance) | ![Red] |
| Algorithms which require coordination between instances, benefit from “a global view” | ![Red] |
| Large scale tables with relatively infrequent updates (ARP,..) | ![Red] |
| Software/Algorithm for tightly coupled homogeneous environments | ![Red] |
| Controlled/tightly-managed Environments | ![Red] |
| Rapid response to Topology Changes: Efficient “plain vanilla” Layer-3-style forwarding | ![Red] |
| Rapid response to data-plane events / packet forwarding | ![Red] |
| Simplicity of Control- and Data-Plane Integration** | ![Red] |

**Past experience (e.g. PSTN AIN, Softswitches/IMS, SBC): CP/DP split requires complex protocols between CP and DP.

Considerations for Control-Plane Evolution

Decision making feedback loops – Example: Packet forwarding control

- Event/State Source: E.g. Link state change
- State Distribution: E.g. Link state distribution
- State Processing: E.g. Topology computation
- Decision Making: E.g. Route computation
- Action Taking: E.g. FIB programming
Packet Forwarding Decision Making

An obvious observation

- Network devices delivering line rate forwarding performance need to take forwarding decisions as quickly as packets arrive
  - In case a forwarding rule exists, apply rule “at line rate”
  - In case a forwarding rule does not exist
    Buffer the packet and create a rule (or ask someone else to create a rule)
    How much buffer do you have?
    Drop the packet
    Flood the packet

<table>
<thead>
<tr>
<th>Interframe Gap (IFG)</th>
<th>Standard (96 bit times)</th>
<th>Minimum on reception*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ethernet</td>
<td>9.6us</td>
<td>4.7us</td>
</tr>
<tr>
<td>Fast Ethernet</td>
<td>0.96us</td>
<td>Not defined</td>
</tr>
<tr>
<td>Gigabit Ethernet</td>
<td>0.096us</td>
<td>0.064us</td>
</tr>
<tr>
<td>10 Gigabit Ethernet</td>
<td>0.0096us</td>
<td>0.0047us</td>
</tr>
</tbody>
</table>

*IFG shrinking is allowed to cope with variable network delays, clock tolerances, added preamble bits
Forwarding decision making
Pro-Active and Re-Active Mode

- **Pro-Active**: Compute rules upfront and install in the devices
  - Typical behavior of a router

- **Re-Active**: Decide forwarding rules once packets arrive
  - Send first (or first few) packets of a flow to a decision making entity (“Controller”) which creates a forwarding rule for the flow – and optionally performs forwarding on the first (or first few packets)
  - Bridges “kind-of” do this: Derive forwarding tables from packet MAC addresses: Requires hardware based learning

- **Combinations**
  - Default to pro-active, leverage re-active for certain exceptions (e.g. few, long living flows)
Reactive Mode: Considerations

- Decision making delay and associated buffering requirements
- Flow-setup / Flow-teardown latency
- Scale
  - Bandwidth/packet forwarding requirements between forwarding entity and decision making entity
  - Total number of flows, feasibility of rule aggregation
Consistency and Availability
New Trade-offs in specific/constrained domains?

- Consistency and Availability Trade-off
  - Typical databases are better at Consistency than Availability, wide area databases cannot have both
  - Trade-off typically domain specific

- Trade-off between concurrency, performance and consistency
  - Strict consistency can come at cost in update/read latency and lower throughput
  - Trade-off typically domain specific

- Classic network protocols
  - Designed for eventual consistency and partial failure; solutions with different scope (i.e. IGP, BGP, ..)

“CAP Theorem”: You can have at most two of these properties for any shared data system (Eric Brewer, UC Berkeley)
Global vs. Local Optimum

- Packet forwarding today solved as a distributed state problem
  - Optimal solution for shortest path (wrt/ link metrics)

- Distributed algorithms sometimes only locate local optimum, not a global optimum
  - Central view/control eases finding a global optimum
  - Frequent change of optimization parameters and algorithms used is easier if only done on a single system, rather than in a distributed environment
    - OSPF RFC: 100+ pages on state distribution, only a few on the actual Dijkstra algorithm
  - Compute vs. Bandwidth trade-off: Centralization optimizes for CPU utilization, but requires additional bandwidth to get to a decision
  - Speed vs. Accuracy: Often you go for a suboptimal but quick decision, compared to an optimal, but slow decision (“great late” vs. “acceptable fast”)
“Learning” vs. “Managed”

- Different Administration Paradigms in Networking
  - Learning systems: Derive decisions from observed behavior/events
  - Managed systems: Decisions driven through the management/orchestration system

- No one-size-fits all
  - “Federal Model” – central entities defining slowly evolving constraints, combined with quick local, sometimes suboptimal, decision making
Open Network Environment Qualities

Agents and Controllers:
*The OpenFlow Protocol*
OpenFlow

- Original Motivation
  - Research community’s desire to be able to experiment with new control paradigms

- Base Assumption
  - Providing reasonable abstractions for control requires the control system topology to be decoupled from the physical network topology (as in the top-down approach)
    Starting point: Data-Plane abstraction: Separate control plane from the devices that implement data plane

- OpenFlow was designed to facilitate separation of control and data planes in a standardized way

- Current spec is both a device model and a protocol
  - *OpenFlow Device Model*: An abstraction of a network element (switch/router); currently (versions <= 1.3.0) focused on Forwarding Plane Abstraction.
  - *OpenFlow Protocol*: A communications protocol that provides access to the forwarding plane of an OpenFlow Device
Nothing new under the sun
Starting point of Data-Plane Abstraction & Data- and Control Plane separation isn’t new

- **Ipsilon Flow Switching**
  Centralized flow based control, ATM link layer
  GSMP (RFC 3292)

- **AT&T “SDN”**
  Centralized control and provisioning of SDH/TDM networks

- A similar thing happened in TDM voice to VOIP transition
  Softswitch → Controller
  Media gateway → Switch
  H.248 → Device interface

- **IETF ForCES WG**
  Separation of control and data planes
  RFC 3746 (and many others)

- **GMPLS, MPLS-TP**

- **PBB-TE**

- **Multiple Cisco product examples, e.g.**
  Wireless LAN Controller (WLC) – APs
  Nexus 1000V (VSM – VEM)
  Remember RSM (7200 on a stick with Catalyst 5000 as dataplane)?
OpenFlow

Basics

- **OpenFlow Components**
  - *Application Layer Protocol*: OF-Protocol
  - *Device Model*: OF-Device Model (abstraction of a device with Ethernet interfaces and a set of forwarding capabilities)
  - *Transport Protocol*: Connection between OF-Controller and OF-Device*

- **Observation:**
  - OF-Controller and OF-Device need pre-established IP-connectivity

* TLS, TCP – OF 1.3.0 introduces auxiliary connections, which can use TCP, TLS, DTLS, or UDP.

Source: OpenFlow 1.3.0 specification, figure 1
OF Processing Pipeline

**OF 1.0 model**
(single lookup)

**OF 1.1 and beyond model**
(multiple lookups)

1. Find highest-priority matching flow entry
2. Apply instructions:
   i. Modify packet & update match fields (apply actions instruction)
   ii. Update action set (clear actions and/or write actions instructions)
   iii. Update metadata
3. Send match data and action set to next table

Source: OpenFlow 1.3.0 specification, figure 2
## OpenFlow Table

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Match Fields</strong></td>
<td>(ingress port, packet header, metadata from previous table)</td>
</tr>
<tr>
<td><strong>Priority</strong></td>
<td>(matching precedence of flow entry)</td>
</tr>
<tr>
<td><strong>Counters</strong></td>
<td>(matching packets)</td>
</tr>
<tr>
<td><strong>Instructions</strong></td>
<td>(modify action set, pipeline processing)</td>
</tr>
<tr>
<td><strong>Timeouts</strong></td>
<td>(flow expiry)</td>
</tr>
<tr>
<td><strong>Cookie</strong></td>
<td>(opaque data chosen by controller)</td>
</tr>
</tbody>
</table>
Packet Flow through an OpenFlow Switch

Source: OpenFlow 1.3.0 specification, figure 3
# Required Match Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OXM_OF_IN_PORT</td>
<td>Ingress port. This may be a physical or switch-defined logical port.</td>
</tr>
<tr>
<td>OXM_OF_ETH_DST</td>
<td>Ethernet source address. Can use arbitrary bitmask</td>
</tr>
<tr>
<td>OXM_OF_ETH_SRC</td>
<td>Ethernet destination address. Can use arbitrary bitmask</td>
</tr>
<tr>
<td>OXM_OF_ETH_TYPE</td>
<td>Ethernet type of the OpenFlow packet payload, after VLAN tags.</td>
</tr>
<tr>
<td>OXM_OF_IP_PROTO</td>
<td>IPv4 or IPv6 protocol number</td>
</tr>
<tr>
<td>OXM_OF_IPV4_SRC</td>
<td>IPv4 source address. Can use subnet mask or arbitrary bitmask</td>
</tr>
<tr>
<td>OXM_OF_IPV4_DST</td>
<td>IPv4 destination address. Can use subnet mask or arbitrary bitmask</td>
</tr>
<tr>
<td>OXM_OF_IPV6_SRC</td>
<td>IPv6 source address. Can use subnet mask or arbitrary bitmask</td>
</tr>
<tr>
<td>OXM_OF_IPV6_DST</td>
<td>IPv6 destination address. Can use subnet mask or arbitrary bitmask</td>
</tr>
<tr>
<td>OXM_OF_TCP_SRC</td>
<td>TCP source port</td>
</tr>
<tr>
<td>OXM_OF_TCP_DST</td>
<td>TCP destination port</td>
</tr>
<tr>
<td>OXM_OF_UDP_SRC</td>
<td>UDP source port</td>
</tr>
<tr>
<td>OXM_OF_UDP_DST</td>
<td>UDP destination port</td>
</tr>
</tbody>
</table>
OF Match Fields

- OFPXMT_OFB_IN_PORT = 0, /* Switch input port. */
- OFPXMT_OFB_IN_PHY_PORT = 1, /* Switch physical input port. */
- OFPXMT_OFB_METADATA = 2, /* Metadata passed between tables. */
- OFPXMT_OFB_ETH_DST = 3, /* Ethernet destination address. */
- OFPXMT_OFB_ETH_SRC = 4, /* Ethernet source address. */
- OFPXMT_OFB_ETH_TYPE = 5, /* Ethernet frame type. */
- OFPXMT_OFB_VLAN_VID = 6, /* VLAN id. */
- OFPXMT_OFB_VLAN_PCP = 7, /* VLAN priority. */
- OFPXMT_OFB_IP_DSCP = 8, /* IP DSCP (6 bits in ToS field). */
- OFPXMT_OFB_IP_PROTO = 10, /* IP protocol. */
- OFPXMT_OFB_TCP_SRC = 13, /* TCP source port. */
- OFPXMT_OFB_TCP_DST = 14, /* TCP destination port. */
- OFPXMT_OFB_UDP_SRC = 15, /* UDP source port. */
- OFPXMT_OFB_UDP_DST = 16, /* UDP destination port. */
- OFPXMT_OFB_SCTP_SRC = 17, /* SCTP source port. */
- OFPXMT_OFB_SCTP_DST = 18, /* SCTP destination port. */
- OFPXMT_OFB_ICMPV4_TYPE = 19, /* ICMP type. */
- OFPXMT_OFB_ICMPV4_CODE = 20, /* ICMP code. */
- OFPXMT_OFB_ARP_OP = 21, /* ARP opcode. */
- OFPXMT_OFB_ARP_SPA = 22, /* ARP source IPv4 address. */
- OFPXMT_OFB_ARP_TPA = 23, /* ARP target IPv4 address. */
- OFPXMT_OFB_ARP_SHA = 24, /* ARP source hardware address. */
- OFPXMT_OFB_ARP_THA = 25, /* ARP target hardware address. */
- OFPXMT_OFB_IPV6_SRC = 26, /* IPv6 source address. */
- OFPXMT_OFB_IPV6_DST = 27, /* IPv6 destination address. */
- OFPXMT_OFB_IPV6_FLABEL = 28, /* IPv6 Flow Label */
- OFPXMT_OFB_ICMPV6_TYPE = 29, /* ICMPv6 type. */
- OFPXMT_OFB_ICMPV6_CODE = 30, /* ICMPv6 code. */
- OFPXMT_OFB_IPV6 ND_TARGET = 31, /* Target address for ND. */
- OFPXMT_OFB_IPV6 ND_SLL = 32, /* Source link-layer for ND. */
- OFPXMT_OFB_IPV6 ND_TLL = 33, /* Target link-layer for ND. */
- OFPXMT_OFB_MPLS_LABEL = 34, /* MPLS label. */
- OFPXMT_OFB_MPLS_TC = 35, /* MPLS TC. */
- OFPXMT_OFB_MPLS_BOS = 36, /* MPLS BoS bit. */
- OFPXMT_OFB_PBB_ISID = 37, /* PBB I-SID. */
- OFPXMT_OFB_TUNNEL_ID = 38, /* Logical Port Metadata. */
- OFPXMT_OFB_IPV6_EXTHDR = 39, /* IPv6 Extension Header pseudo-field */
OpenFlow Actions

- Output
- Set-Queue* (for QoS)
- Drop
- Group
- Push-Tag/Pop-Tag*
- Set-Field* (e.g. VLAN)
- Change-TTL*

*Optional
OpenFlow Ports
Physical Ports, Logical Ports, Reserved Ports

- Physical Ports == Ethernet Hardware Interfaces
- Logical Ports == ports which are not directly associated with hardware interfaces (tunnels, loopback interfaces, link-aggregation groups)
  - Can include packet encapsulation. Logical ports can have metadata called “Tunnel-ID” associated with them
- Reserved Ports
  - ALL (all ports of the switch)
  - CONTROLLER (represents the control channel with the OF-controller)
  - TABLE (start of the OF-pipeline)
  - IN_PORT (packet ingress port)
  - ANY (wildcard port)
  - LOCAL* (local networking or management stack of the switch)
  - NORMAL* (forward to the non-OF part of the switch)
  - FLOOD*

* Optional
OpenFlow Ports
Simplified View

IN_PORT

CONTROLLER “port”

Physical Port

Logical Port (representing a VLAN)

Logical Port (representing a VLAN)

Logical Port (representing link aggregation group)

LOCAL “Port”

NORMAL “Port”

“Classic Switch part”

“OF-Switch part”

TABLE

“Classic Switch part”

“OF-Switch part”
OpenFlow Ports
CONTROLLER port and NORMAL “port”

**CONTROLLER**
- Forward packets to Controller
- For “reactive” mode of operation
- Considerations
  - Latency for decision making
  - Bandwidth between OF-switch and OF-controller
  - Speed at which rules can be installed/removed

**NORMAL**
- More of a concept than a real “port”: Hand packets to “classic” part of the switch
- Forwarding operation in the classic part is TBD
  - Xconnect?
  - L2-Bridge (use Dest-MAC to forward packet to o/if)?
  - L3-Route (requires L3-next hop info as meta-data from OF, or rely on classic routing protocol)?
Hybrid Model

- One criticism of OpenFlow
  - OpenFlow is making all switches dumb, it requires complete re-implementation of entire control plane in the logically centralized controller (due to OpenFlow being a protocol)

- Hybrid Model acknowledges a more generic approach:
  Re-architect the control plane architecture *where needed*
  - Keep existing control planes on network devices and evolve/complement them
  - Drive definition of additional modules and associated abstractions

- Hybrid Model Concerns include
  - Reconciliation of state required in case multiple modules can create competing decisions (e.g. using the RIB)
  - Potentially requires the OpenFlow device model to evolve and to include additional abstractions
A Couple Of Hybrid Switch Use Cases

- Installing ephemeral routes in the RIB
  - Install routes in RIB subject to admin distance or …
  - Moral equivalent of static routes, but dynamic
  - May require changes to the OF protocol/model

- Edge classification
  - Use OF to install **ephemeral** classifiers **at the edge**
  - Moral equivalent of … ‘ip set next-hop <addr>’ (PBR)
  - Use case: Service Engineered Paths/Service Wires
    - Program switch edge classifiers to select set of {MPLS, GRE, …} tunnels
    - Core remains the same

- Service Chaining
Programmable Service Chains

- Basic Use Cases
  - Endpoints vs. In-line services
  - Composite Services / Service Chaining
  - Flow Routing
    - Fine vs. Coarse Grained Flows
    - Filtering vs. Routing
    - Placement vs. Topology
    - Addressing vs. Flows

- Additional Use Cases to be considered
  - CDN, Optical xconnect,…
ONF Hybrid WG

▪ Goal
  – Explore and document the requirements for a hybrid programmable forwarding plane ("OF controls a subset of all flows"): Will allow definition of required OF protocol extensions
    “Hybrid Switch” and “Hybrid Network”
  – Allows the installed base of switches and routers to be utilized effectively while allowing OF deployments to commence
  – Allows deployment scenarios in which only a subset of the devices are OpenFlow-enabled.

▪ Focus
  – Use-cases for integrating OpenFlow programmed state in existing network and service architectures
  – “Ships in the Night” architecture
  – Will also investigate: “Integrated Architecture”
Hybrid Switch: Ships in the Night vs. Integrated

“Ships-in-the-Night”

• A subset of ports controlled by OF, another subset controlled by router’s native CP – physical resources are partitioned
• Some level of integration: “OF_NORMAL”:
  • Implementer free to define what “normal” is
  • May or may not be what router normally does

“Integrated”

• Use OF for feature definition – augment the native control plane
• No longer partitioning of resources
• Can operate at different abstraction levels (low-level like OF1.0 or higher level)
OpenFlow Versions

Dec 31, 2009
OF 1.0
- Single Table
- L2, IPv4 focused matching

Feb 28, 2011
OF 1.1
- Multiple Tables
- MPLS, VLAN matching
- Groups: {Any-,Multi-}cast
- ECMP

Dec 5, 2011
OF 1.2
- IPv6
- Flexible-length matching

April 19, 2012
OF 1.3.0
- 802.1ah PBB
- Multiple parallel channels between Switch and Controller
OpenFlow Evolution
Making OF functionally complete

- Topics of ongoing work
  - High availability model for device and controller (state re-sync etc.)
  - Security model (granular access control)
  - L3-forwarding model
  - Enhanced Statistics
  - Management infrastructure (evolution of OF-CONFIG)
  - Testing and certification framework
  - Hybrid device/network deployment capability (Hybrid WG)

- Longer term futures (OF “2.0”)
  - Develop a FPDL (Forwarding Plane Description Language)?
OpenFlow Evolution Challenge

Device Model: Getting the right abstraction is hard

Example Device Models

- "Forwarder, Exact match lookup"
- "Generic Forwarder"
- "Forwarder with Packet Queuing/QoS"
- "Switch/Router"
- "VPN Switch/Router"

"Future" / "hybrid" OpenFlow versions?

- OpenFlow 1.1
- OpenFlow 1.2/1.3

Model complexity increases with set of use-cases/solutions covered

- Location-based Services
- Mobility
- Traffic Optimization (WAAS...)
- Address Translation
- Security (Firewall...)
- Identity
- Network Virtualization/Topology Control
- Enhanced Traffic/Event reporting
- Network Graph Traversal / Routing
- Encapsulation Control
- QoS Control
- Device capability exchange
- Basic statistics
- Packet Filtering
- Packet Forwarding Control – LPM lookup
- Packet Forwarding Control – Exact match lookup

...
Open Network Environment Qualities

Agents and Controllers: Leveraging the right Agent-Controller pair for the job

- Programmatic APIs
- Agents and Controllers
- Network Infrastructure Virtualization
Agents and Controllers – Task specific sets

- Networking leverages a great breath of Agents and Controllers
  - Data-Collection (Netflow), Configuration (SNMP, Netconf, CAPWAP, Diameter, Radius), Service Assurance (IPSLA), Path Computation (PCE), Performance Routing (PfR),..
  - Many routing schemes use “controllers”, they elect the controller within the protocol (and using administrative settings of priority, we can ensure a specific controller is elected)
Devices, Controllers – and APIs

- APIs represent abstractions at different layers – complementing each other
  - Device-layer, Network-layer etc.
  - Devices can deliver network level abstractions and APIs as well (e.g. link state topology)
  - Common, consistent API, different scopes

Example abstractions and associated APIs delivered through controllers
Example abstractions delivered by individual device

Device level abstractions
Service/Network level abstractions

Common consistent set of APIs

API

Device layer
Network layer

Forwarding Policy, QoS
Packet Data-Path Access
Network Topology
Service Placement
Service Path
Devices, Controllers – and APIs

Utility APIs
Area APIs
PIN APIs
Element APIs

BGP
RADIUS
Netconf
HSRP
EIGRP
DHCP
ICMP
MPLS
etc

Device/Domain Intermediation
Federated Controllers

Deployments typically combine Device-APIs, device delivered Network-APIs, and controller delivered Network APIs for a particular solution

Example: Data-Center Interconnect across two providers with granular traffic forwarding control
Controllers as Aggregation and Decision Points

Example: Network Positioning System (NPS)

- Computes the location of and distance between endpoints
  - Caching and replication are vital to optimization of network traffic. Distribution paradigms efficiency is augmented by dynamic mechanisms that locate (and determine distance to) services and data in order to optimize infrastructure resources utilization
  - Example: need to locate the nearest copy of a movie or the closest instance of a service among several available resources

- NPS/Proximity leverages network layer and Policy information.
  - Extended to other information sources such as: state & performance and Geo-location

Aggarwal et al. show that when the p2p overlay topology is network aware, it is highly correlated with the underlying network topology; the nodes within an AS form a dense cluster, with only a few connections going to nodes in other AS.

Example: NPS
ALTO as the API

- Example:
  Request Reply Model: Address Ranking
  - Which targets in a given list of IP addresses are the closest to a particular query source (e.g.: user IP address) ?
  - Simple location & distance request by application to network

- Leverages IETF ALTO (Application Layer Traffic Optimization) API
  - An API, through which an application can request guidance from the network, here: for locating or placing services
  - Preserves confidentiality between layers: No need to know atomic topology details
Example: Topology Exposure: Multi-Area IGP
ALTO server exposes multi-area IGP topology

- ALTO server needs to know all areas topology
  - Manually crafting of “IGP peering” topology is tedious and error prone

- Approach:
  - Advertize Link-State Information in BGP
  - draft-gredler-bgp-te
Full-Duplex Access across all Network Planes
Example: SP Network Model and Technology Context

Examples of evolving technologies:

- Stateful PCEP: e.g. draft-crabbe-pce-stateful-pce
- NPS/ALTO: e.g. draft-ietf-alto-protocol
- CDN Internconection
- onePK APIs
- Protocols: NetConf/YANG, NetFlow, PCEP, Diameter, OF, ...
- Topology exposure, e.g. BGP-LS
- virtual Service Appliances/Gateways
- Service Chaining, vPath
- OF-Hybrid devices
- TE enhancements: draft-previdi-isis-metric-extensions
- GENAPP: draft-isis-genapp-extensions
- MPLS-TP

Service Control and Admin

NPS
Multi-Layer PCE, iOverlay
CDNI

Service Orchestration

Topologies, Statistics

Paths, Tunnels

Service Wires – Service Chains/Topologies
IP/MPLS tunnels – Layer-3 Topologies
Wavelengths – Transport Topologies

Examples of evolving technologies:
Open Network Environment Qualities

Network Infrastructure Virtualization

- Programmatic APIs
- Agents and Controllers
- Network Infrastructure Virtualization
“In computing, **virtualization** is the creation of a virtual (rather than actual) version of something, such as a hardware platform, operating system, storage device, or network resources.”

http://en.wikipedia.org/wiki/Virtualization
Network Abstractions support Virtualization
Blurring the lines between physical and virtual entities – networks and services

Common Abstractions and common APIs across physical and virtual network elements

- Virtual Overlay Networks
  - custom endpoint addressing (e.g. for simple endpoint mobility)
  - custom topologies/segmentation
  - custom service chains
    Example: vPATH

- Virtual Service Appliances/Gateways
  - VSG, vWAAS

Map ‘n Encap approaches to allow for flexible overlays and “identity” and “location” addresses:
- **L2-transport**: FabricPath, 802.1ah
- **IP-transport**: VXLAN, OTV, (L2-)LISP (all use the same frame format)
- **MPLS-transport**: (PBB-)VPLS, (PBB-)EVPN
Virtual Overlay Networks

Example: Virtual Overlay Networks and Services with Nexus 1000V

- Large scale L2 domains: Tens of thousands of virtual ports
- Common APIs
  - Incl. OpenStack Quantum API’s for orchestration
- Scalable DC segmentation and addressing
  - VXLAN
- Virtual service appliances and service chaining/traffic steering
  - VSG (cloud-ready security), vWAAS (application acceleration), vPATH
- Multi-hypervisor platform support: ESX, Hyper-V, OpenSource Hypervisors
- Physical and Virtual: VXLAN to VLAN Gateway
Complementing classic VPN technologies
Network Partitioning, a.k.a. “Slicing”

- VPN (L3VPN, L2VPN) technologies combine
  - Network Partitioning/Segmentation
  - Packet Forwarding Control (Control plane)

- “Slicing” refers to Network Partitioning only, i.e. no assumptions on the control plane made
  - Slices fully isolated (one slice not effecting resources and operation of other slices)
  - Several existing technologies incorporate “slicing concepts”, e.g.
    - PBB-TE – network partitioned based on I-SID/VLANs (one partition controlled by STP, another one through a NMS)
    - MPLS-TP

- “Network slicing manager”
  - Slicing manager defines/administers slices and maintains view of all slices in the network
  - Users only see their “slice” – can be used e.g. as sandbox network for a given Dept/Developer
Virtualization: Network Partitioning

Example: Network Slicing for Research Environments

- **Business Problem**
  - University desires to “slice” the network into multiple partitions:
    - Production network – classic control plane
    - Several research networks – experimentation with new control algorithms, programs etc.

- **Solution**
  - Network Slicing Manager partitions the network based on e.g. ports or VLANs
    - Provides northbound interfaces, incl. OpenFlow (Flowvisor-like)
    - Effects of a particular control function of a partition/slice limited to that partition/slice
Open Network Environments and Software Defined Networks
Some History…
The early SDN architecture approach

- Define target distribution architecture upfront
  - Assume that suitable abstractions can be found

- Approach
  - Data-plane abstraction as starting point
  - Network global view/state abstraction (*semantic free*)
  - Control-programs (deal with task-specific semantics)
Observations on early SDN architecture view

- Finding the right (set of) device abstraction is hard
  - Too simple and you can’t control important switch features, too complex and it becomes un-implementable
  - One size fits all probably impossible (see earlier experiences with e.g. H.248,…)
  - Missing standardized definitions for northbound API/interface of the controller; missing abstractions/APIs for Control Modules

- Data-plane abstraction closely linked to OpenFlow protocol
  - Protocol nature inspires all control software to be removed from the network device, i.e. implemented on a server (and be re-written)
  - In order to run the OpenFlow protocol the network device needs to implement a control plane independent from what is controlled through OpenFlow (using OpenFlow only for an overlay network with the underlying transport still using the normal control plane could be an escape out of this chicken-and-egg situation)
Defining Generic Abstractions is Hard

Example: Network Graph Abstraction Layer

- Network visualization
  - Loose timing and accuracy requirements

- Service/Load placement
  - Longer term heuristic algorithms used for service placement, thus limited accuracy required

- Forwarding: Generic Routing
  - Eventual consistency between forwarding and control state (TTL for temporary loop protection)
  - Sub-second convergence time: Fast reaction to all occurring events

- Forwarding: Generic Bridging
  - Strong consistency between forwarding and control state required (no loop protection in dataplane)
  - Sub-second convergence time: Fast reaction to all occurring events

Forwarding focused abstractions: Consider operational/debugging aspects
Evolve the early SDN Model…
... acknowledge the need for diverse abstractions
Programmatic Network Control and the Gartner Hype Cycle

Technology Readiness Phases

Phase 1

Academic research ("OpenFlow")

Phase 2

Broad interest from technical community + use cases ("SDN")

Phase 3

Value Proposition for mass market understood

Entering Phase 3...

Source: John Vrionis, LightSpeed Venture, opennetsummit.org/talks/ONS2012/vrionis-wed-closing.pdf
Open Network Environment for SDN

Applications Management Orchestration

Utility/Area/PIN APIs (e.g. ALTO, ...)

Service/Network Abstractions

Network-Policy
Service-Path (incl. PCE, ...)
Location/Topology (incl. ALTO, BGP-LS, ...)

Utility/Area/PIN/Element APIs
(ex. ConnectedApps, PCEP, OpenFlow, LISP-MS, OpenStack Quantum API, ...)

APIs, Protocols, Encapsulations (e.g. BGP, IS-IS, ...)

Physical Network Device

Virtual Network Device
Open Network Environment for SDN
Focus: Network/Service Abstractions and associated APIs

Applications Management Orchestration

Orchestration

Dynamic Service Chains
Multi-POD Network containers
MediaNet Delivery Network
Cloud Cache Network
...

Service/Network Abstractions

Network-Policy
Service-Path (incl. PCE, …)
Location/Topology (incl. ALTO, BGP-LS, …)

Physical Network Device

Virtual Network Device
Open Network Environment
Focus: Network/Service Abstractions and associated APIs
Open Network Environment for SDN
Focus: Device-level abstractions and associated APIs
Open Network Environment: Standards Context
Orchestrating Efforts across multiple SDOs/Fora, including IETF, IEEE, ONF, ITU, …

Programmatic APIs

Open Network Environment

Agents and Controllers

Network Infrastructure Virtualization

Legend: 1) Cisco chaired/co-chaired
2) Key contributions from Cisco

- Network Configuration (NETCONF WG)
- Application Layer Traffic Optimization (ALTO WG)
- Content Delivery Networks Interconnection (CDNI WG)
- Extensible Messaging and Presence Protocol (XMPP WG)
- Software Driven Networks (SDNP BOF)
- Infrastructure-to-application information exposure (I2AEX BOF)

- OpenFlow Device Configuration (ONF Config WG)
  - Path Computation Control (PCE WG)
  - Forwarding & Control Separation (FORCES WG)

- OF Hybrid Devices/Networks (Hybrid WG)
- OF Protocol Extensions (Extensibility WG)
- OF Futures: “FMOD/OF2.0” (Futures group)

- SG13 – Future Networks
  - Y.FNsdn: SDN Framework
  - Y.FNsdn-fm: Requirements formal specification and verification methods for SDN

- Network Virtualization Overlays (NVO3 WG)
- L2VPN WG
- L3VPN WG
- Locator/ID Separation Protocol (LISP WG)
- Pseudowire Emulation Edge to Edge (PWE3 WG)

- IEEE
- 802.1aq Shortest Path Briding
- 802.1Qbp Equal Cost Multiple Paths
- 802.1Qbg – Edge Virtual Bridging
- 802.1BR – Bridge Port Extension

- SG13 – Future Networks (Y.Fnvirtreq - Requirement of network virtualization)
Cisco Vision: Exposing The Entire Network Value
Programmatic Control across Multiple Network Planes

Program Policies for Optimized Experience

Application Developer Environment

Analysis and Monitoring, Performance and Security

Network Elements and Abstraction

Any Object
- Switch/Router
- ASIC
- Network Fabric
- Compute

Any Service
- Cloud
- Collaboration
- Video
- Security
- Mobility

Any Layer
- L1-7
- Control/Data Plane
- Hardware/Software
- ASICs/OS

Harvest Network Intelligence
Open Network Environment – Summary
The Industry’s Broadest Approach to Programmatic Access to the Network

- Evolutionary step for networking: Complement/evolve the Network Control Plane where needed
- Centered around delivering open, programmable environment for real-world use cases
  - No one-size-fits-all
  - Cisco will support Network Virtualization, APIs and Agents/Controllers
  - Joint evolution with industry and academia
- Technology-agnostic
  - Not predicated on a particular technology or standard
  - Draw from Cisco technologies and industry standards
- Delivered as incremental functionality
  - Many customers will use hybrid implementations
  - Build upon existing infrastructure with investment protection

Open Network Environment
www.cisco.com/go/one

onePK
www.cisco.com/go/onepk
www.cisco.com/go/getyourbuildon
Complete Your Online Session Evaluation

- Give us your feedback and you could win fabulous prizes. Winners announced daily.
- Receive 20 Passport points for each session evaluation you complete.
- Complete your session evaluation online now (open a browser through our wireless network to access our portal) or visit one of the Internet stations throughout the Convention Center.

Don’t forget to activate your Cisco Live Virtual account for access to all session material, communities, and on-demand and live activities throughout the year. Activate your account at the Cisco booth in the World of Solutions or visit www.ciscolive.com
Final Thoughts

- Get hands-on experience with the Walk-in Labs located in World of Solutions, booth 1042
- Come see demos of many key solutions and products in the main Cisco booth 2924
- Visit www.ciscoLive365.com after the event for updated PDFs, on-demand session videos, networking, and more!
- Follow Cisco Live! using social media:
  - Facebook: https://www.facebook.com/ciscoliveus
  - Twitter: https://twitter.com/#!/CiscoLive
  - LinkedIn Group: http://linkd.in/CiscoLI