Software Defined Networking : Primer

Muhammad Moinur Rahman

History of Networking

- Blackbox networking equipments
- Big name companies building switching/routing devices
- Includes Proprietary/OEM Silicon Chip
- Wrapped up with a closed source Operating System (e.g. A desktop PC with MS Windows and MS Office)

Disadvantages of Current Scenario

Technology was not designed keeping today in mind

- Massive Scalability
- Multi Tenant Networks
- Virtualization
- Cloud Computing
- Mobility (Users/Devices/VM)

Disadvantages of Current Scenario(Contd)

Protocols are Box Centric; Not Fabric Centric

- Difficult to configure correctly(consistency)
- Difficult to add new features(upgrades)
- Difficult to debug(look at all devices)

Disadvantages of Current Scenario(Contd)

Closed Systems (Vendor Hardware)

- Stuck with given interfaces (CLI, SNMP, etc.)
- Hard to meaningfully collaborate
- Vendors hesitant to open up
- No way to add new features by yourself

ANSWER: Software Defined Networking

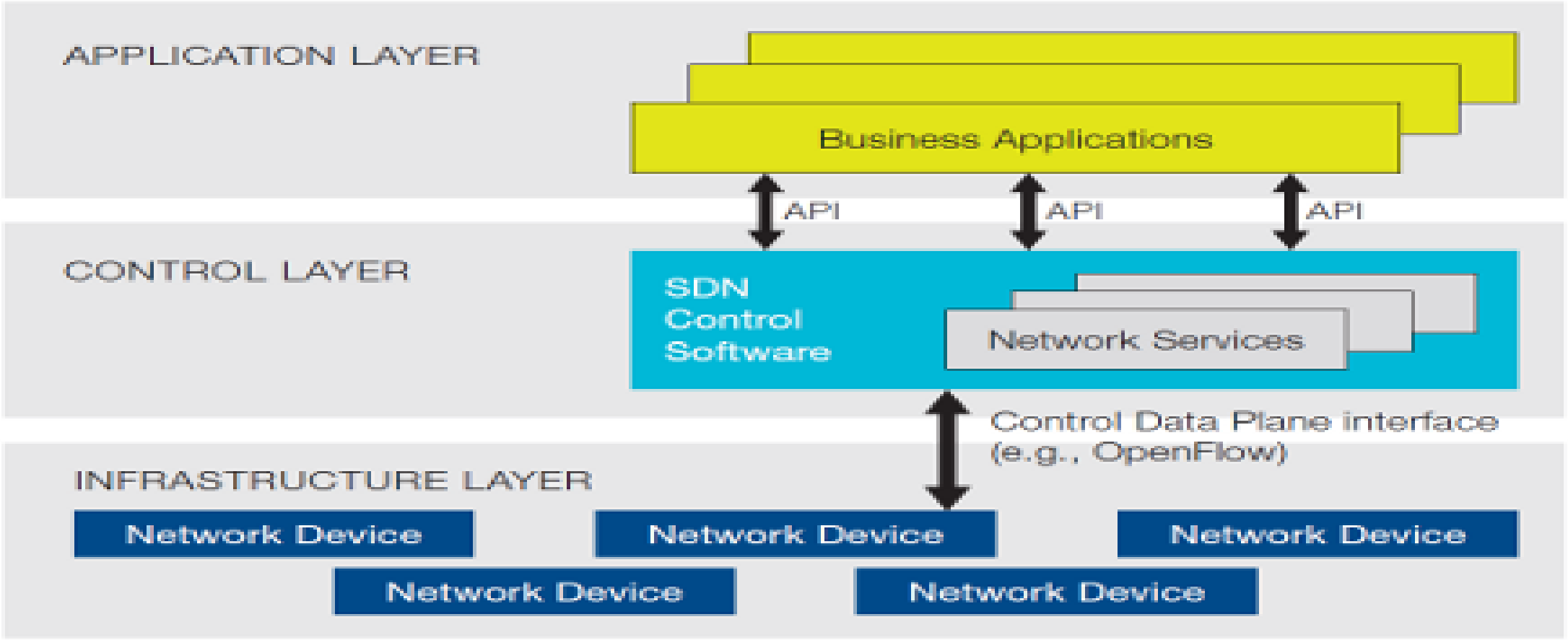
What is SDN?

SDN is a framework to allow network administrators to automatically and dynamically manage and control a large number of network devices, services, topology, traffic paths, and packet handling (quality of service) policies using high-level languages and APIs. Management includes provisioning, operating, monitoring, optimizing, and managing FCAPS (fault, configuration, accounting, performance, and security) in a multi-tenant environment.

Networking Planes

- Data Plane
  - Carries Network User Traffic
- Control Panel
  - Carried Signalling Traffic
- Management Panel
  - Carries Administrative Traffic

SDN Architecture



Need for SDN

Virtualization

- Use network resource
  - without worrying about where it is physically located
  - how much it is
  - how it is organized

Orchestration

- Should be able to control and manage thousands of devices with one command

Programmable

- Should be able to change behavior on the fly

Dynamic Scaling

- Should be able to change size, quantity, capacity

Need for SDN - (Continued)

- Automation
  - To lower OpEx
  - Minimize manual involvement
  - Troubleshooting
  - Reduce downtime
  - Policy enforcement
  - Provisioning/Re-provisioning/Segmentation of resources
  - Add new workloads, sites, devices, and resources
- Visibility
  - Monitor resources, connectivity

Need for SDN - (Continued)

- Performance

  Optimize network device utilization
  - Traffic engineering/Bandwidth management
  - Capacity optimization/Load balancing
  - High utilization
  - Fast failure handling
- Multi Tenancy

  Tenants need complete control over their
  - Addresses/Topology
  - Routing/Security

Need for SDN (Continued)

Service Integration

Provisioned on demand and placed appropriately on the traffic path

- Load balancers
- Firewalls
- Intrusion Detection Systems (IDS)

Alternative APIs

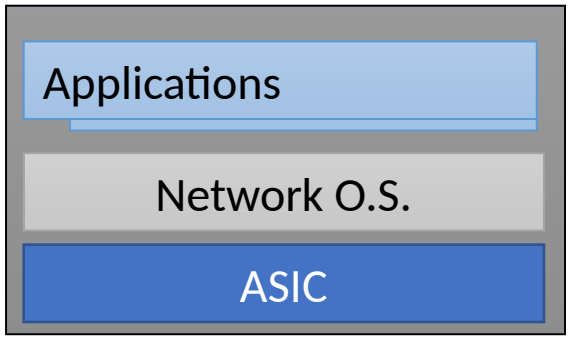
- Southbound APIs: XMPP (Juniper), OnePK (Cisco)
- Northbound APIs: I2RS, I2AEX, ALTO
- Overlay: VxLAN, TRILL, LISP, STT, NVO3, PWE3, L2VPN, L3VPN
- Configuration API: NETCONF
- Controller: PCE, ForCES

History

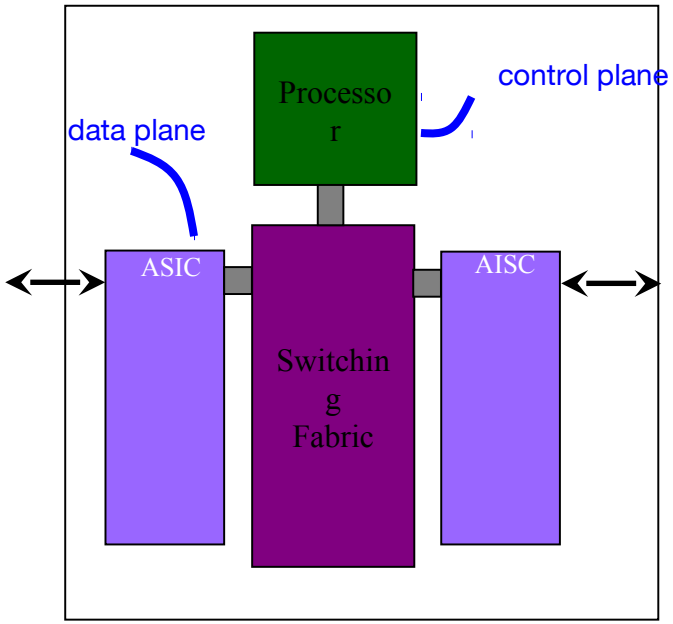
Feb, 2011 - OpenFlow 1.1 Released
Dec, 2011 - OpenFlow 1.2 Released
Feb, 2012 - “Floodlight” Project Announced
Apr, 2012 - Google announces at ONF
Jul, 2012 - Vmware acquires Nicira
Apr, 2013 - “OpenDaylight” Released

Hardware Internals

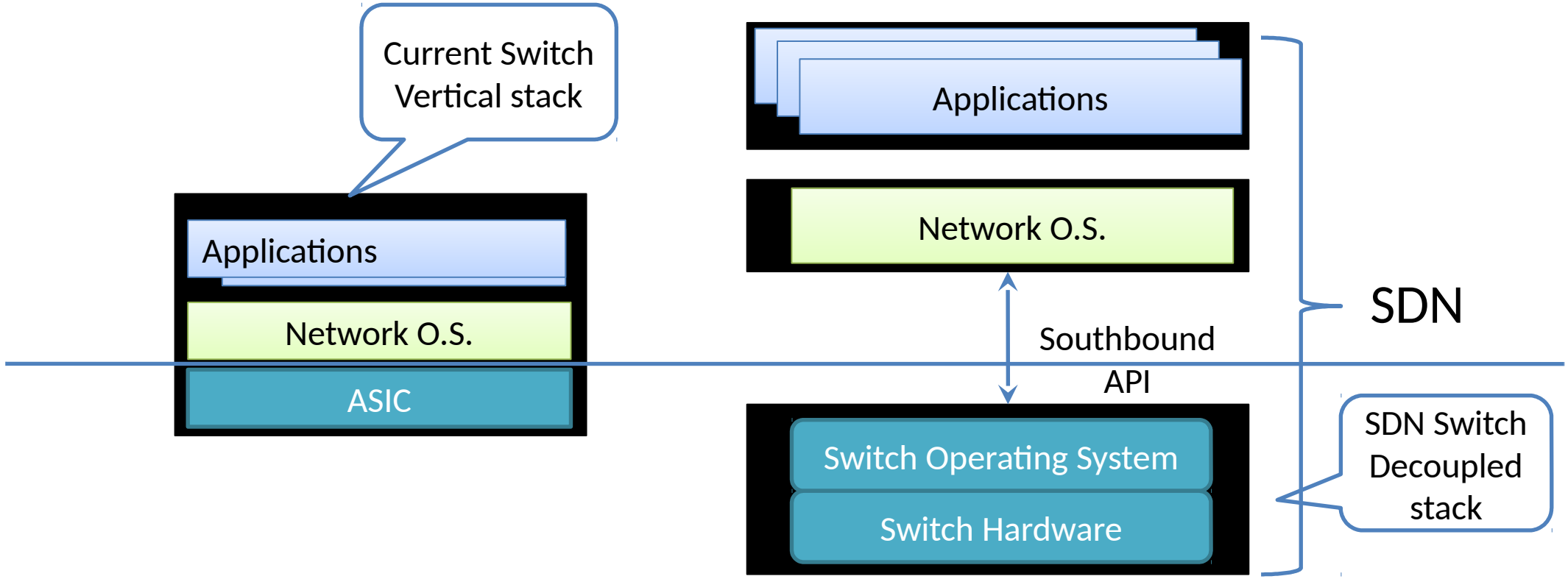
- Logical View of a Switch



- Physical Architecture of a Switch

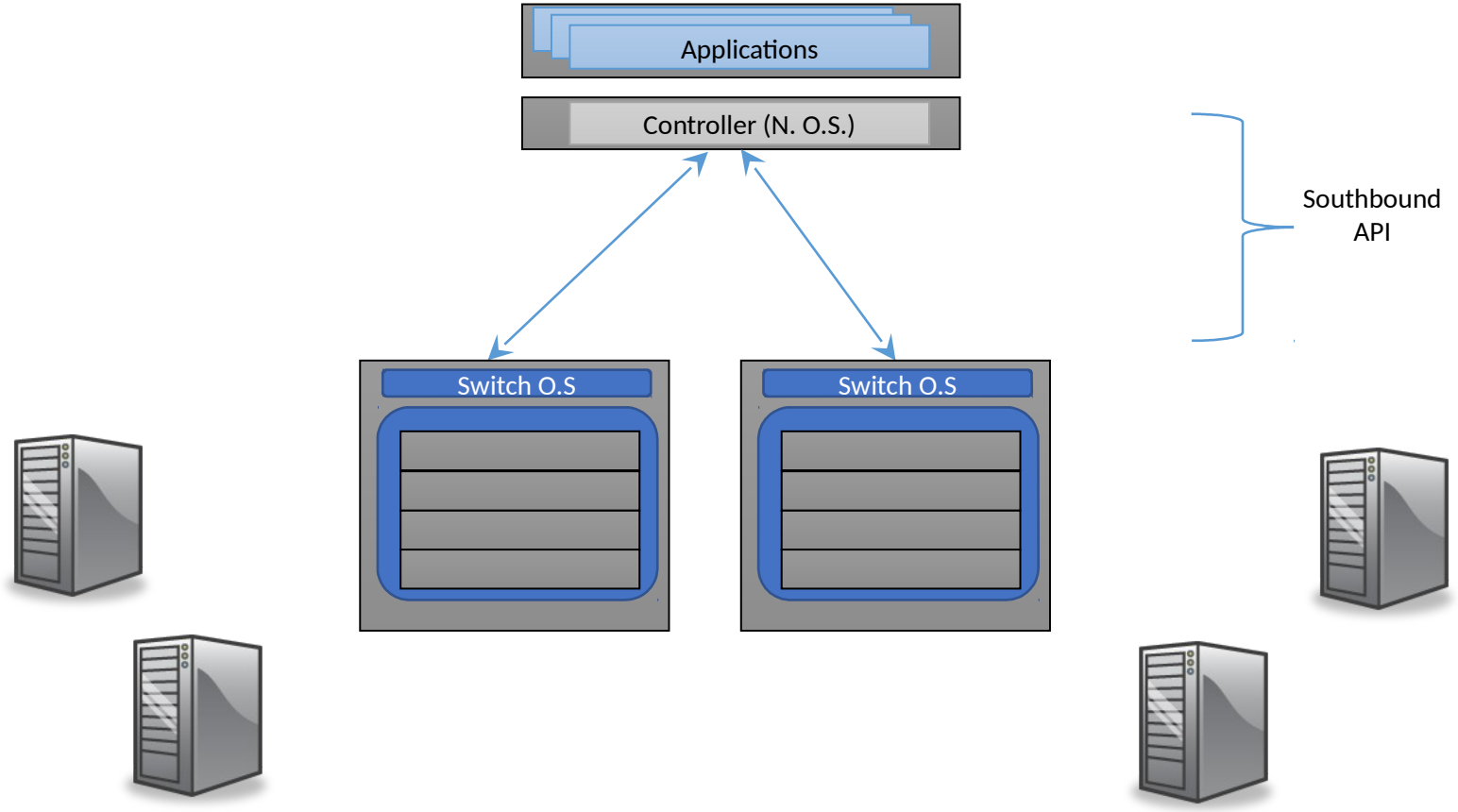


Internals of SDN



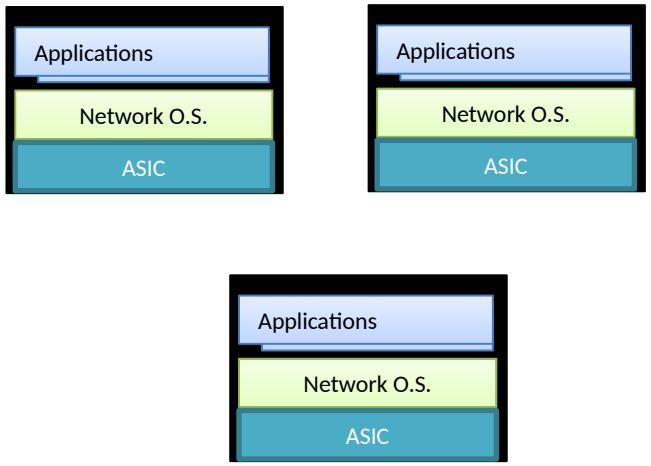
- Southbound API: decouples the switch hardware from control function
  - Data plane from control plane
- Switch Operating System: exposes switch hardware primitives

How SDN Works



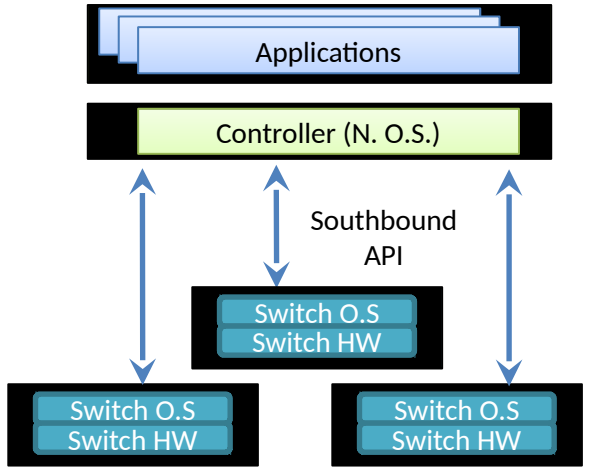
Implications of SDN

Current Networking



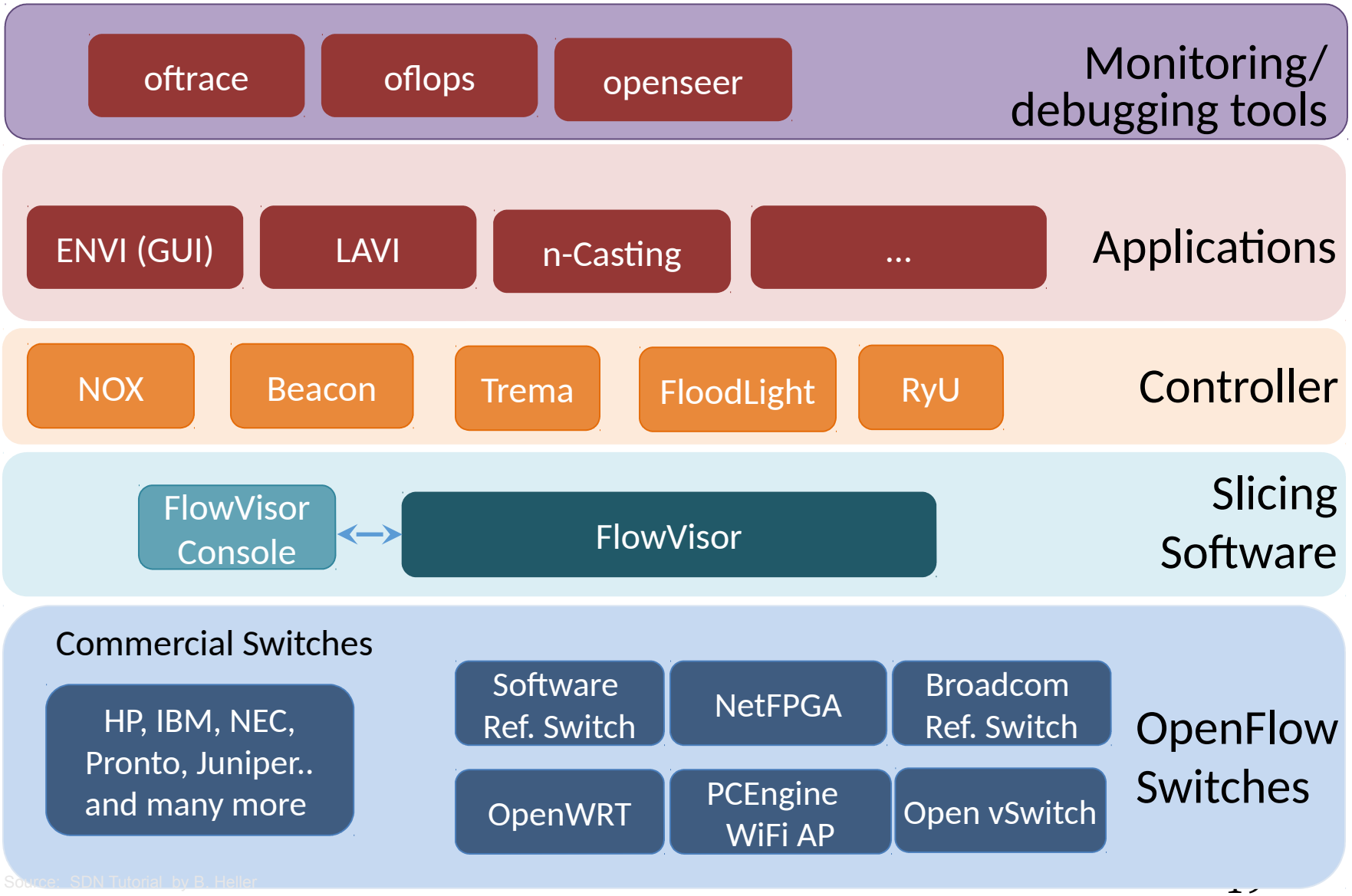
- Distributed protocols
  - Each switch has a brain
  - Hard to achieve optimal solution
- Network configured indirectly
  - Configure protocols
  - Hope protocols converge

SDN Enabled Environment



- Global view of the network
  - Applications can achieve optimal
- Southbound API gives fine grained control over switch
  - Network configured directly
  - Allows automation
  - Allows definition of new interfaces

The SDN Stack



Source: SDN Tutorial by B. Heller
Open Networking Summit, April 2012

Dimensions of SDN Environments: Vendor Devices

Vertical Stacks

- Vendor bundles switch and switch OS
  - Restricted to vendor OS and vendor interface
- Low operational overhead
  - One stop shop

Whitebox Networking

- Vendor provides hardware with no switch OS
- Switch OS provided by third party
  - Flexibility in picking OS
- High operational overhead
  - Must deal with multiple vendors

Dimensions of SDN Environments: Switch Hardware

Virtual: Overlay

- Pure software implementation
  - Assumes programmable virtual switches
  - Run in Hypervisor or in the OS
  - Larger Flow Table entries (more memory and CPU)
- Backward compatible
  - Physical switches run traditional protocols
- Traffic sent in tunnels
  - Lack of visibility into physical network

Physical: Underlay

- Fine grained control and visibility into network
- Assumes specialized hardware
  - Limited Flow Table entries

Dimensions of SDN Environments: Southbound Interface

OpenFlow

- Flexible matching
  - L2, L3, VLAN, MPLS
- Flexible actions
  - Encapsulation: IP-in-IP
  - Address rewriting:
    - IP address
    - Mac address

BGP/XMPP/IS-IS/NetConf

- Limited matching
  - IS-IS: L3
  - BGP+MPLS: L3+MPLS
- Limited actions
  - L3/l2 forwarding
  - Encapsulation

Dimensions of SDN Environments: Controller Types

Modular Controllers

- Application code manipulates forwarding rules
  - E.g. OpenDaylight, Floodlight
- Written in imperative languages
  - Java, C++, Python
- Dominant controller style

High Level Controllers

- Application code specifies declarative policies
  - E.g. Frenetic, McNettle
- Application code is verifiable
  - Amendable to formal verification
- Written in functional languages
  - Nettle, OCamal

Ecosystem

Name	Controller Type	Southbound API	SDN Device	SDN Flavor
Bigswitch	Modular/Floodlight	Openflow 1.3	Whitebox(indigo)	Underlay+Overlay
Juniper	OpenContrail	XMPP/NetCONF/BGP+MPLS	Vertical Stack(Proprietary JunOS)	Overlay
Cisco	Openflow+Proprietary	Openflow 1.3	Vertical Stack(Proprietary IOS/NxOS/IOS-XR)	Underlay+Overlay
Arista	Openflow+Proprietary	Openflow 1.3	Vertical Stack	Underlay
Broadcom	Openflow+Proprietary	Openflow 1.3	Vertical Stack	Underlay
HP	Openflow	Openflow 1.3-1.4	Vertical Stack	Underlay
Dell	Openflow	Openflow 1.3	Vertical Stack	Underlay
FloodLight	Openflow	Openflow 1.0-1.4	Whitebox	Underlay+Overlay
Alcatel	Modular	BGP+MPLS	Vertical Stack	Overlay

OpenFlow

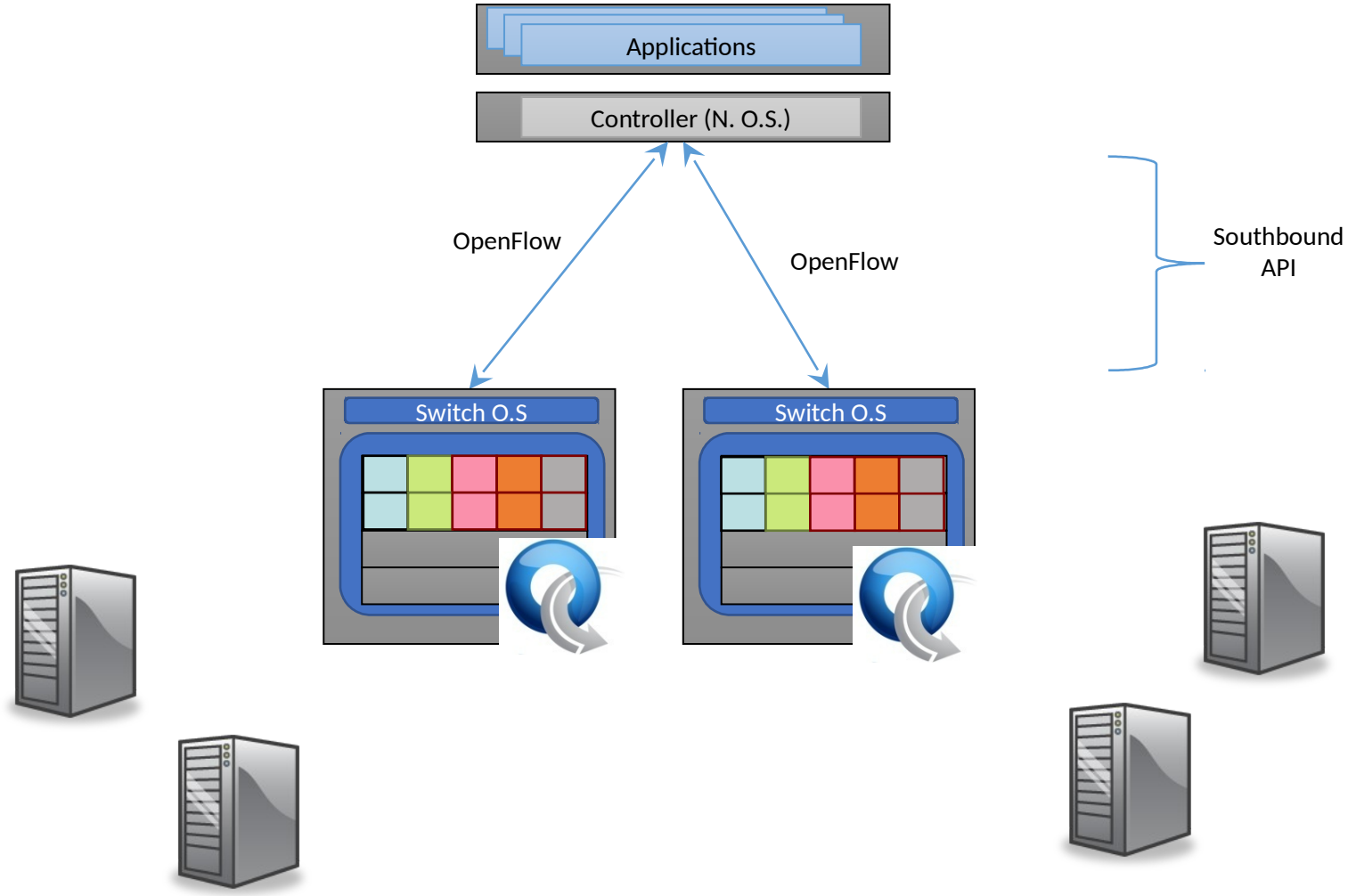
- Developed in Stanford
  - Standardized by Open Networking Foundation (ONF)
  - Current Version 1.4
    - Version implemented by switch vendors: 1.3

- Allows control of underlay + overlay
  - Overlay switches: OpenVSwitch/Indigo-light

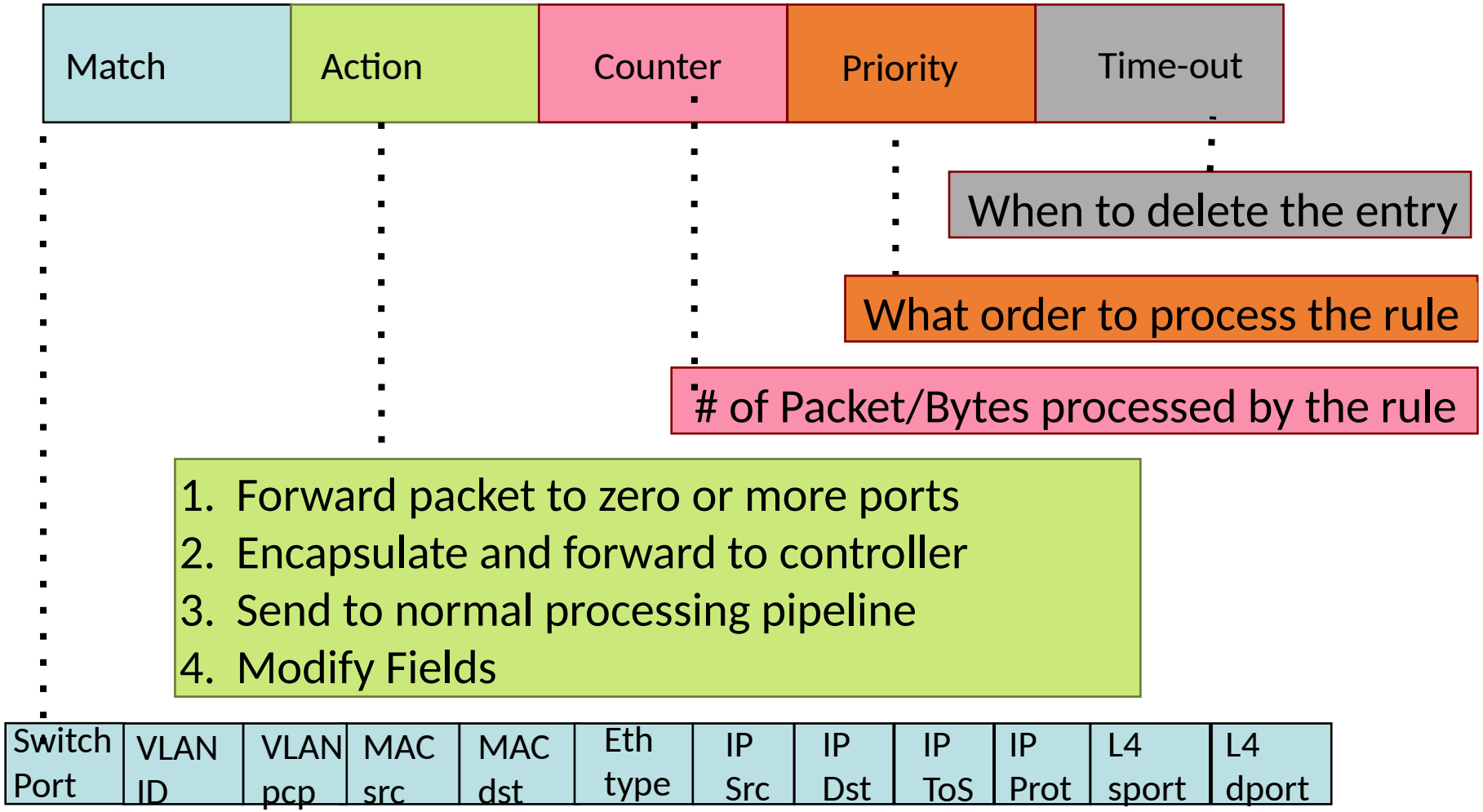
SDN vs OpenFlow

- Leading SDN protocol
- Decouples control and data plane by giving a controller the ability to install flow rules on switches(Bare Metal)
- Hardware or software switches can use OpenFlow
- Spec driven by ONF

How SDN Works: OpenFlow



OpenFlow: Anatomy of a Flow Table Entry



Examples

Switching

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	00:1f:..	*	*	*	*	*	*	*	port6

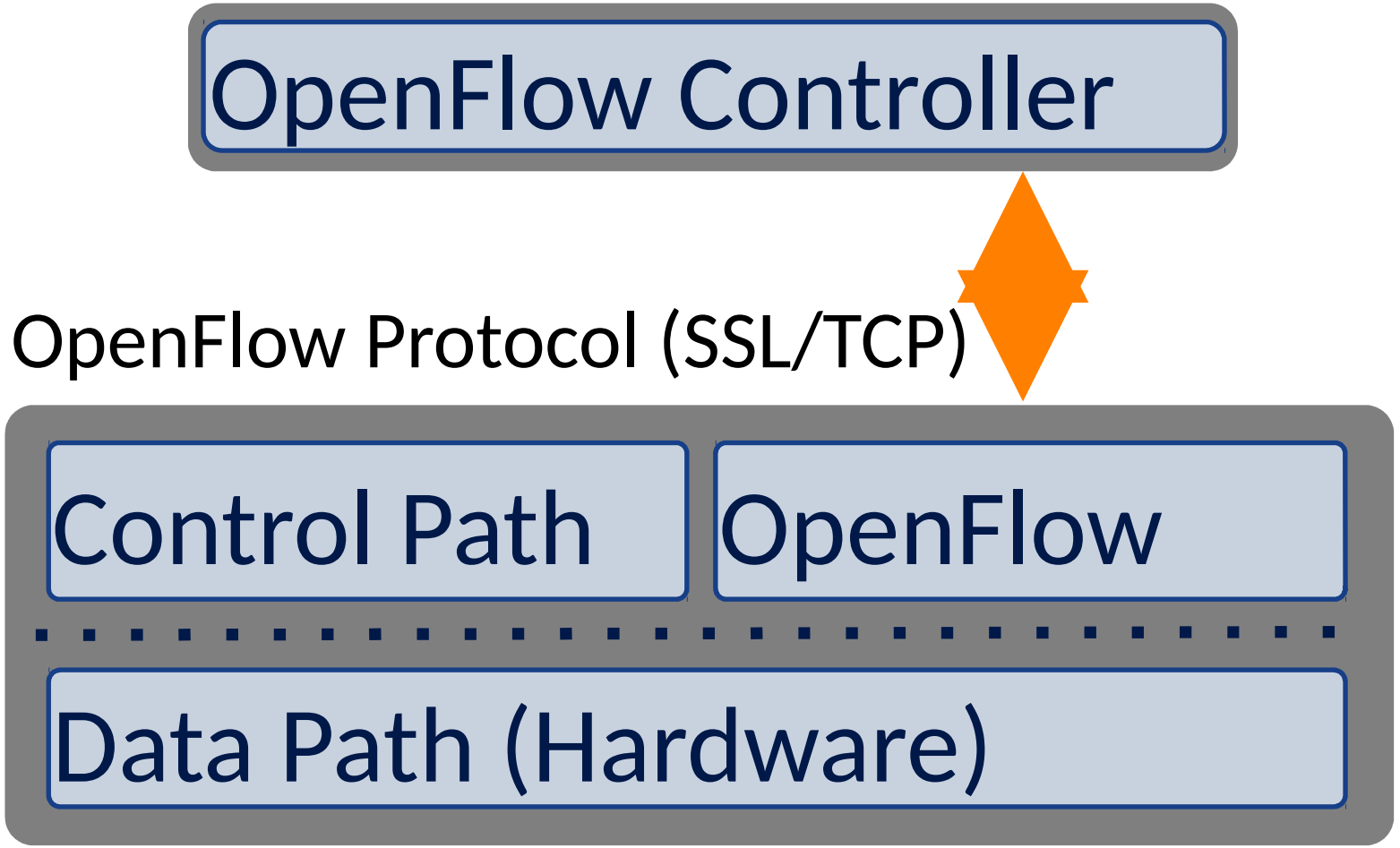
Flow Switching

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
port3	00:20..	00:1f..	0800	vlan1	1.2.3.4	5.6.7.8	4	17264	80	port6

Firewall

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	*	*	*	*	*	*	*	22	drop

OpenFlow: How it works



SDN Components : Hardwares

OpenFlow Compliant (1.0-1.4) Switch

- HP 8200 ZL, 6600, 6200ZL
- Brocade 5400ZL, 3500
- IBM NetIron
- Juniper OCX1100
- Baremetal Switch
- OpenVSwitch

SDN Components : Controllers

OpenFlow Compliant (1.0-1.4) Controller

- POX: (Python) Pox as a general SDN controller that supports OpenFlow. It has a high-level SDN API including a queriable topology graph and support for virtualization.
- IRIS: (Java) a Resursive SDN Openflow Controller created by IRIS Research Team of ETRI.
- MUL: (C) MūL, is an openflow (SDN) controller.
- NOX: (C++/Python) NOX was the first OpenFlow controller.
- Jaxon: (Java) Jaxon is a NOX-dependent Java-based OpenFlow Controller.
- Trema: (C/Ruby) Trema is a full-stack framework for developing OpenFlow controllers in Ruby and C.
- Beacon: (Java) Beacon is a Java-based controller that supports both event-based and threaded operation.
- Floodlight: (Java) The Floodlight controller is Java-based OpenFlow Controller. It was forked from the Beacon controller, originally developed by David Erickson at Stanford.
- Maestro: (Java) Maestro is an OpenFlow "operating system" for orchestrating network control applications.
- NDDI - OESS: OESS is an application to configure and control OpenFlow Enabled switches through a very simple and user friendly User Interface.
- Ryu: (Python) Ryu is an open-sourced Network Operating System (NOS) that supports OpenFlow.
- NodeFlow (JavaScript) NodeFlow is an OpenFlow controller written in pure JavaScript for Node.JS.
- ovs-controller (C) Trivial reference controller packaged with Open vSwitch.

Refereces

1. SDN – The Next Wave of Networking – Siva Valiappan

Questions