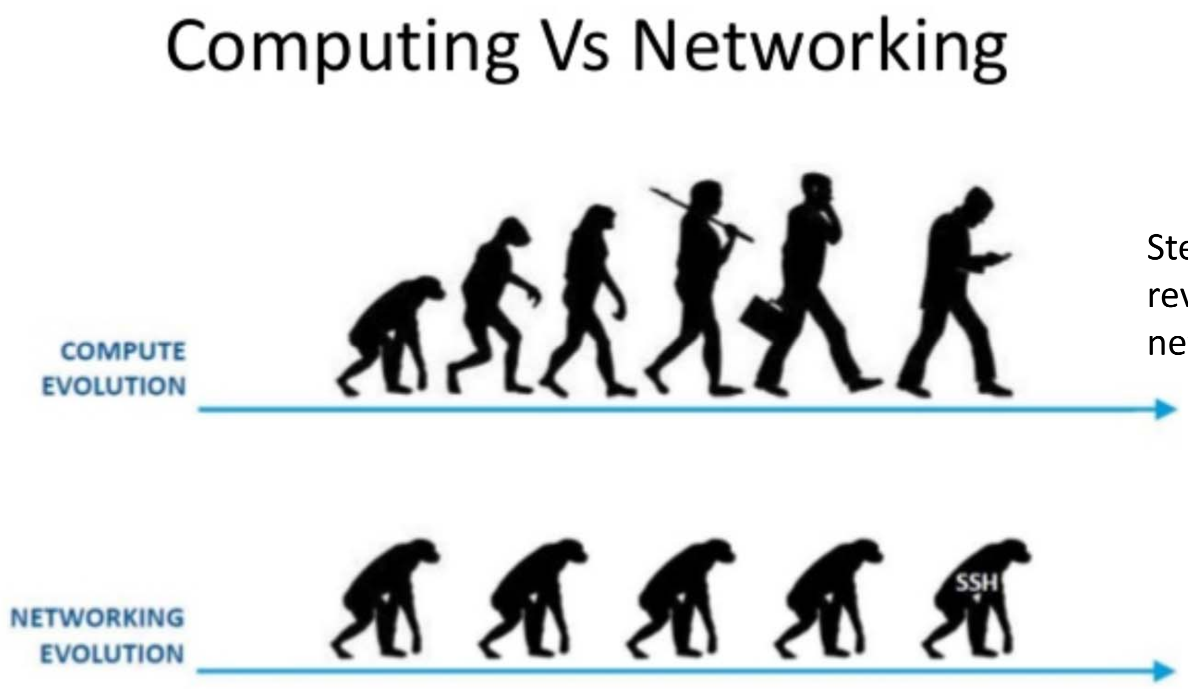
Toward Software defined Networking

Dr. Saeedeh Parsaeefard

Outline

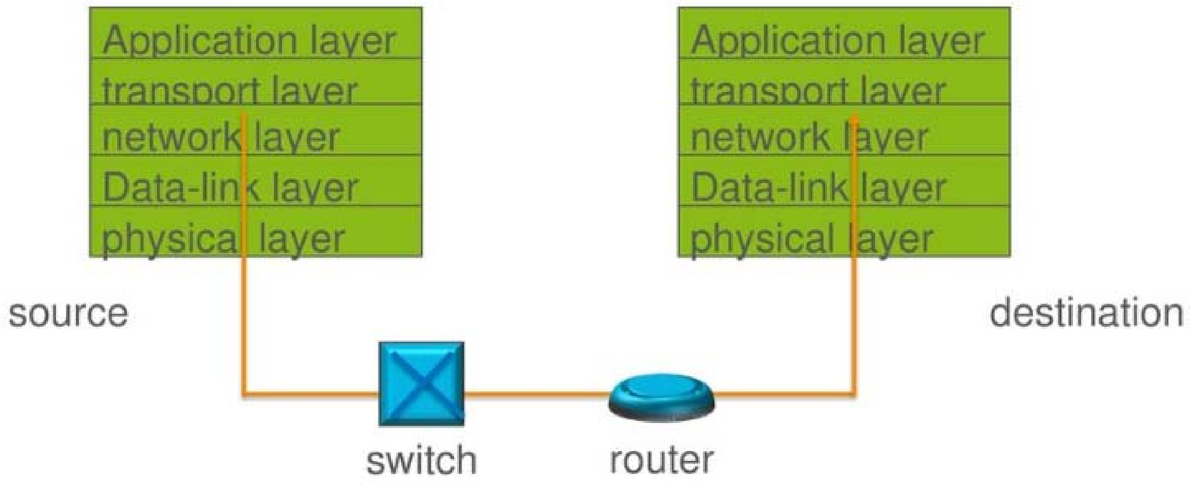
- Learning Objectives
  - Gain awareness about key concepts and principles behind SDN
- Outline
  - Traditional networking
  - Brief history
  - The advent of SDN
  - SDN features, structure and main functionality
  - Some important examples of SDN devices and protocosl
  - SDN future

Why?



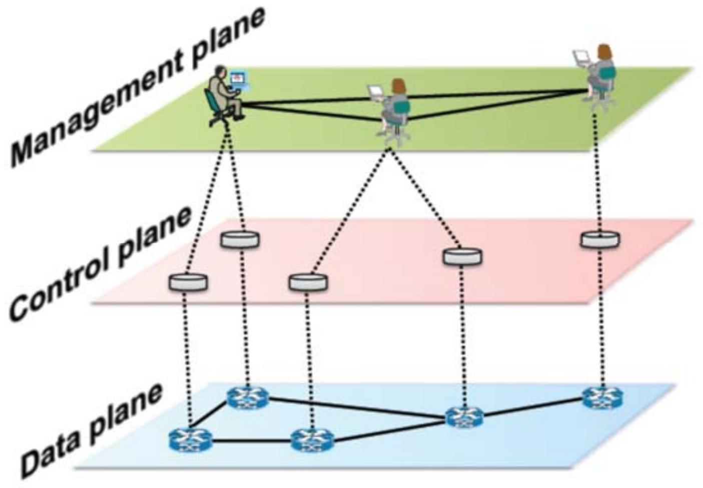
Steps toward revolutionized networking

Overview



Why layer: Good abstraction but overlapping tasks and complicated model for development

Layered View of Networking Functionality



Data Plane

Responsible for forwarding packets

Control Plane

Responsible for defining the behavior of the network

Management Plane

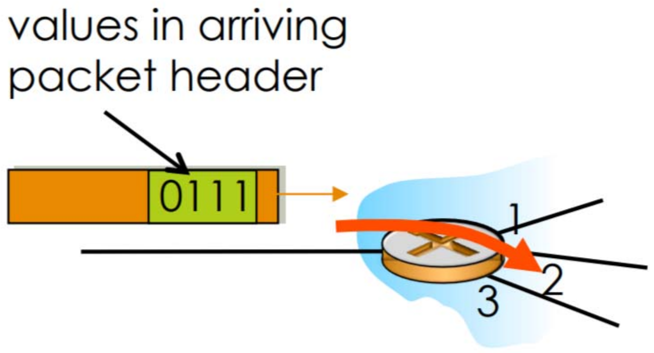
Configures routing protocol, monitors network devices and handle anomalies

Service Plane

Provides services beyond packet forwarding

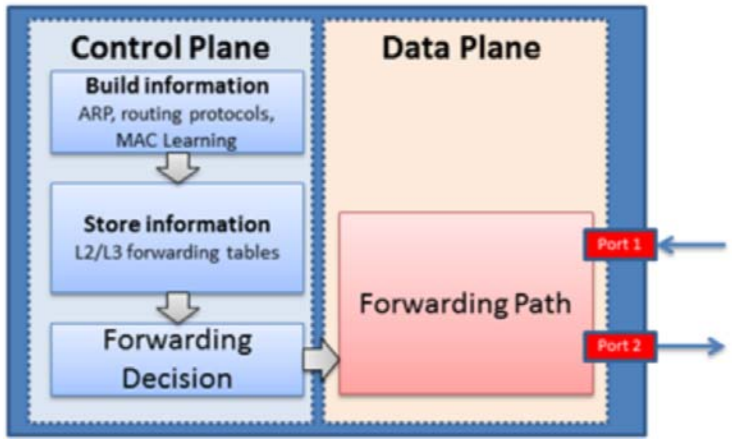
Data Plane

- Responsible for moving packets
  - Local, per-router function
  - Determines how packet arriving on router input port is forwarded to router output port
  - Forwarding function
- Other data plane functions:
  - Packet scheduling and buffering
  - Packet fragmentation
  - Bandwidth metering
  - Access control
  - Packet cloning for multicasting
  - Traffic monitoring



Control Plane

- Defines the network behavior
  - Network-wide logic
  - Determines how packet is routed among routers along end-end path from source host to destination host
  - Route computation function
- Other control plane functions
  - FIB installation in routers
  - Topology discovery
  - Access control list generation
  - QoS enforcement



Service Plane

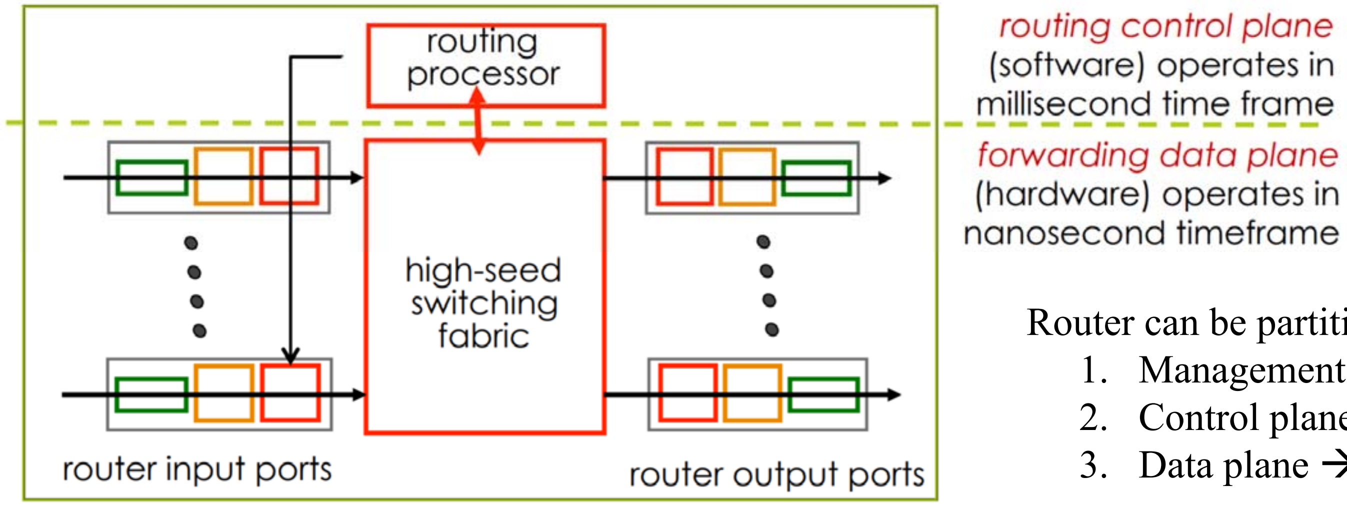
- Service Plane provides services beyond packet forwarding (e.g., Security firewalls, WAN acceleration etc.)
- Services are typically realized by middleboxes
  - Load balancers, Web proxy, Web cache
  - Firewall,
  - Intrusion Detection System (IDS)
  - Intrusion Protection System (IPS)
  - WAN Optimizers, etc.

Management Plane

- Management plane configures routing protocol, and monitors network devices and reacts to anomalies.
  - Responsible for managing **FCAPS**
    - Fault
    - Configuration
    - Accounting
    - Performance
    - Security
  - Provisions resources for network services

Traditional Networking

- Control plane is vertically integrated with data plane
  - Each router runs its own control plane
  - Generic router architecture

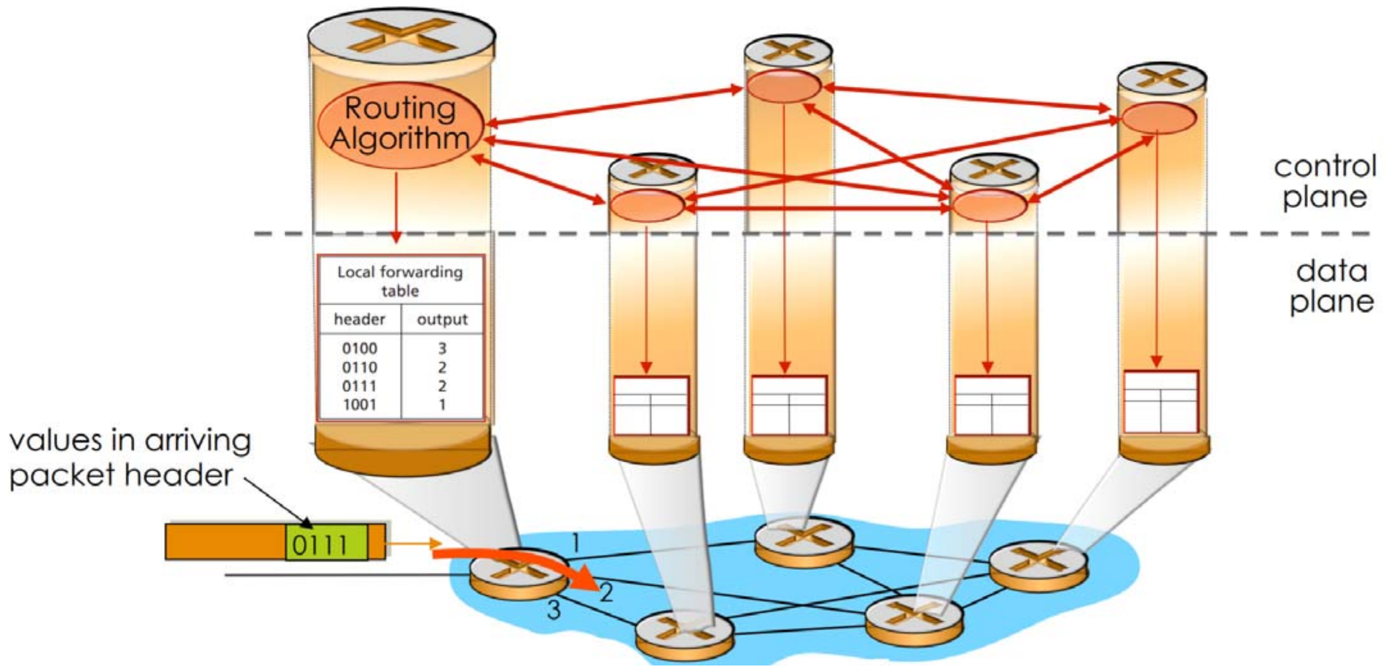

routing processor
high-seed switching fabric
router input ports
router output ports
routing control plane
(software) operates in millisecond time frame
forwarding data plane
(hardware) operates in nanosecond timeframe

Router can be partitioned into three planes

1. Management plane → configuration
2. Control plane → make decision for the route
3. Data plane → data forwarding

Traditional Routing

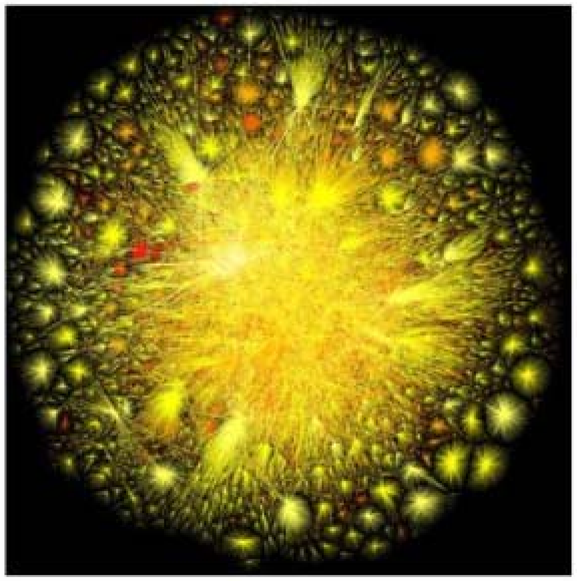
- Traditional routing protocols are distributed
  - Routers exchange connectivity info. with each other
  - Each router builds its own routing table
  - Forwarding tables (FIB) are populated based on information available in the routing table



Traditional Networking

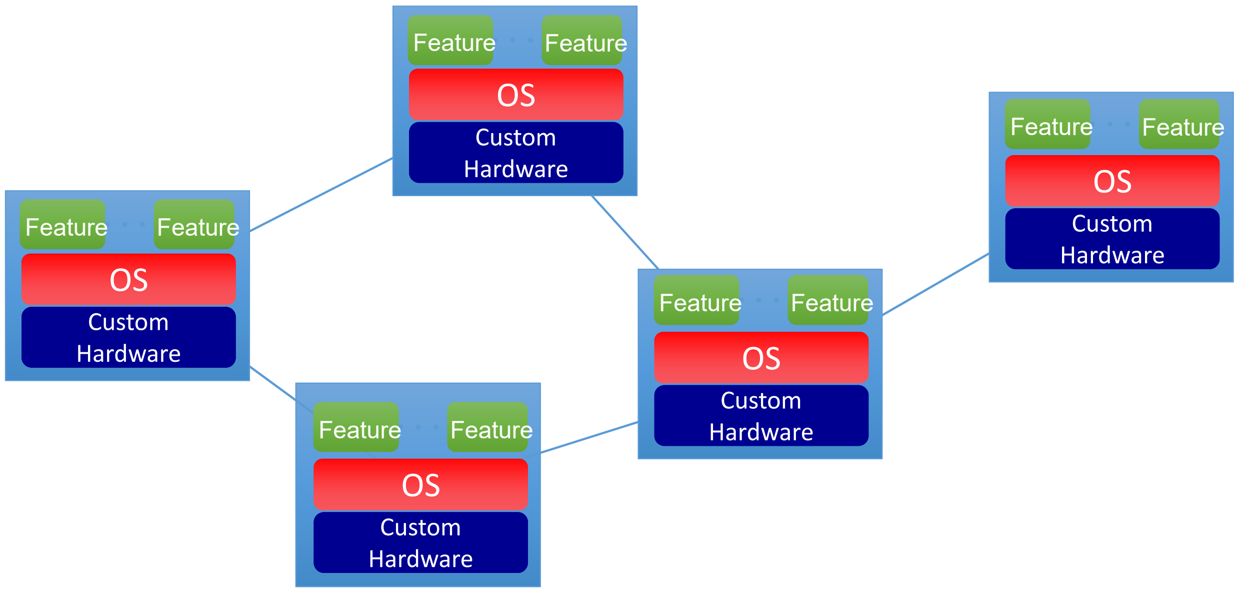
Design principle of IP based networks (Internet)

- Simple
- Intelligent end points
- Distributed controllers
- Scalable

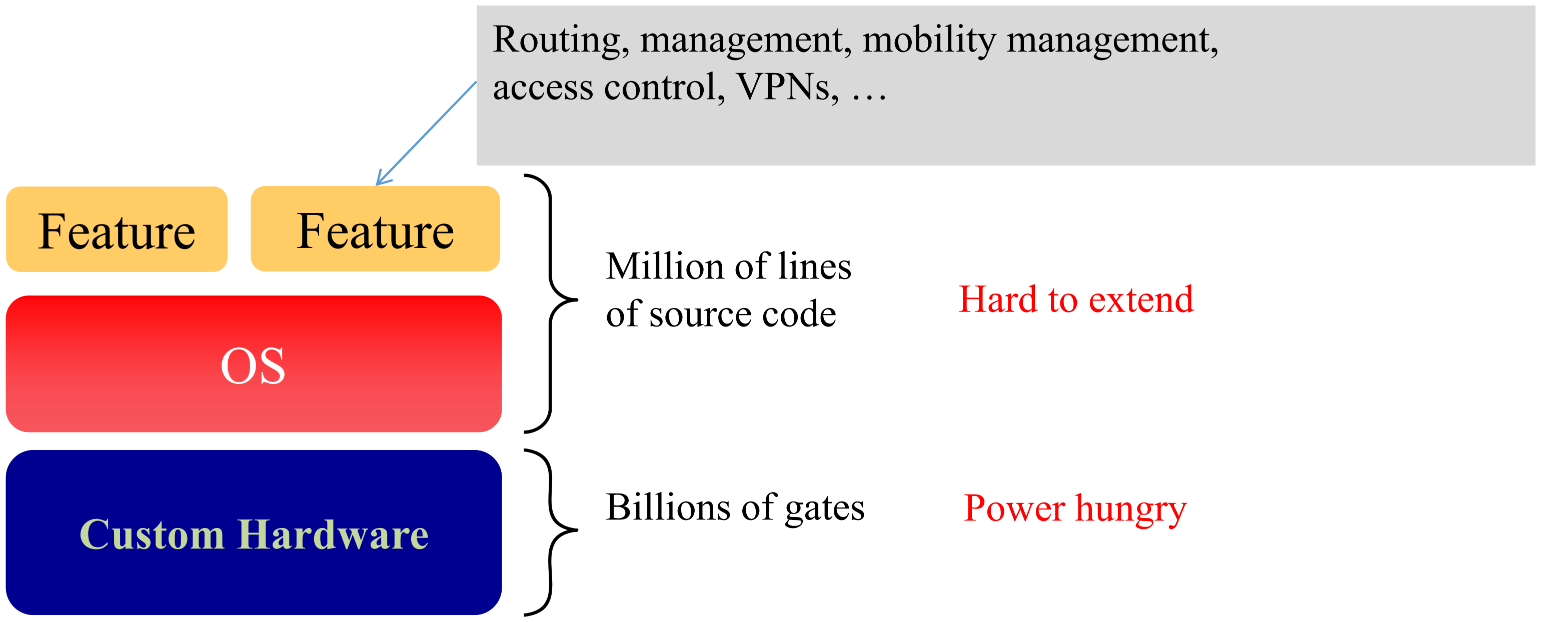


Traditional networking
Distributed Control Plane

- Distributed per-router control plane
  - Vertically integrated router contains switching hardware, runs proprietary implementation of Internet standard protocols (IP) in proprietary router OS (e.g., Cisco IOS)



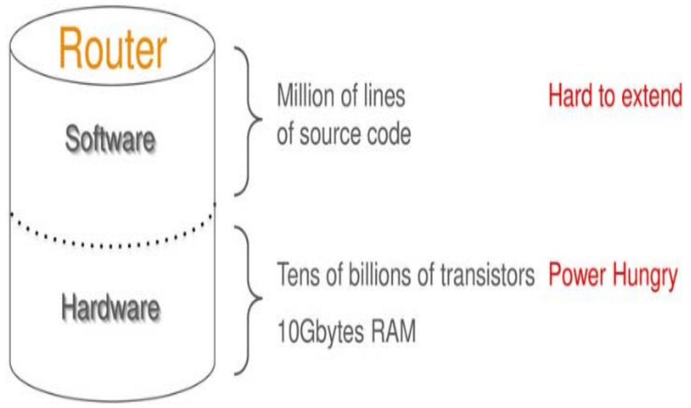
Traditional Network Element


Routing, management, mobility management, access control, VPNs, …
Feature
Feature
OS
Custom Hardware
Million of lines of source code
Hard to extend
Billions of gates
Power hungry

- Vertically integrated, complex, closed, proprietary
- Networking industry with “mainframe” mind-set

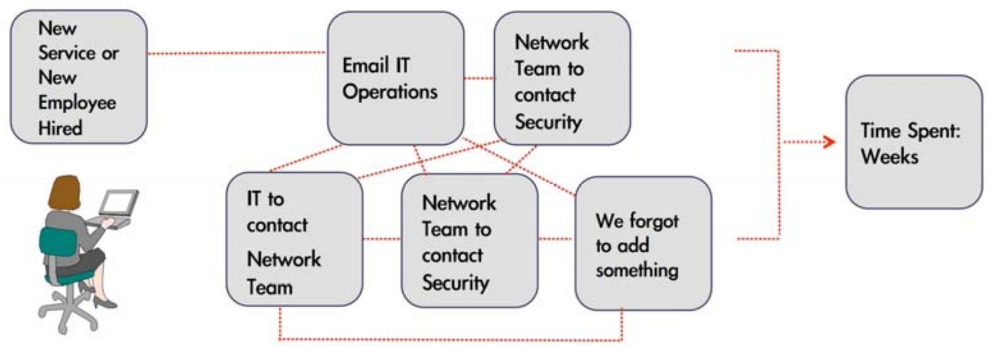
Traditional Networking Issues

- Traditional networks struggles to complex nodes
- Lack of programmability
  - Changes in the network requires manual reconfiguration
  - Error-prone and time consuming
  - Network becomes hard to scale
- Vendor dependency
  - Handful of vendors have monopoly in the market
  - Lack of open interface
  - Software vertically integrated with hardware
- Existing provisioning process is slow, time consuming, human-dependent
- Complex router: Vertically integration with many complex functions
  - Different types of protocols (OSPF, BGP, MPLS), multicast, QoS, Traffic engineering, firewalls


Router
Software
Hardware
Million of lines of source code
Hard to extend
Tens of billions of transistors
Power Hungry
10Gbytes RAM

Traditional Networking Issues

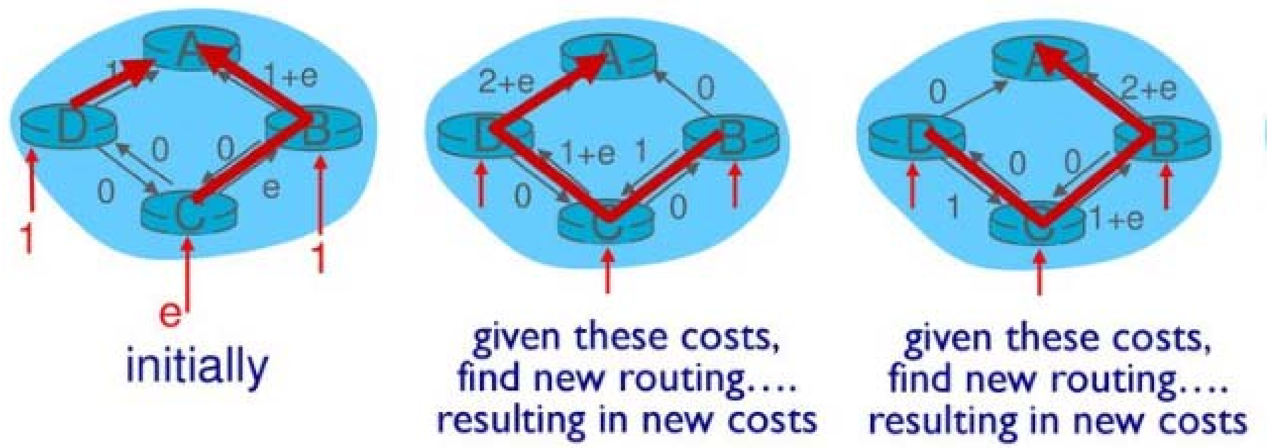
- Less opportunities for innovation
  - Only vendors can write the code
  - Slow time-to-market for new services/features
- High operational cost
  - Human error still causes most outages
- Buggy software
  - Hardware with 20+ million lines of code
  - Cascading failures, vulnerabilities, etc.
  - Complex algorithms



New Service or New Employee Hired
Email IT Operations
Network Team to contact Security
IT to contact Network Team
Network Team to contact Security
We forgot to add something
Time Spent: Weeks

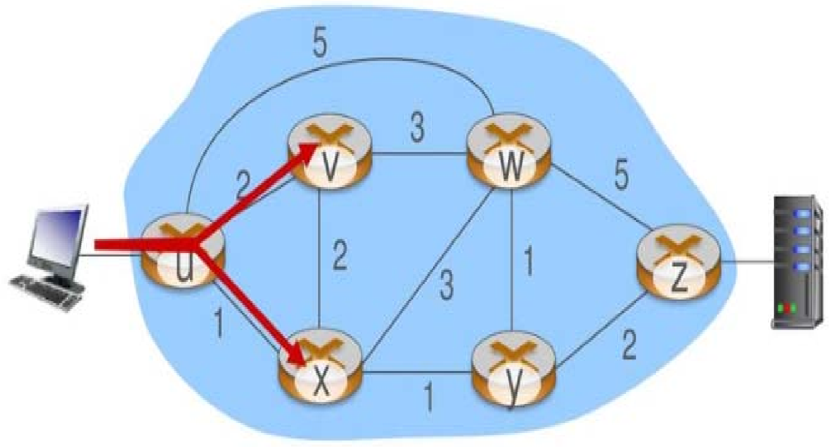
Bugs: Oscillation Problems

Assume that link cost equals the amount of carried traffic

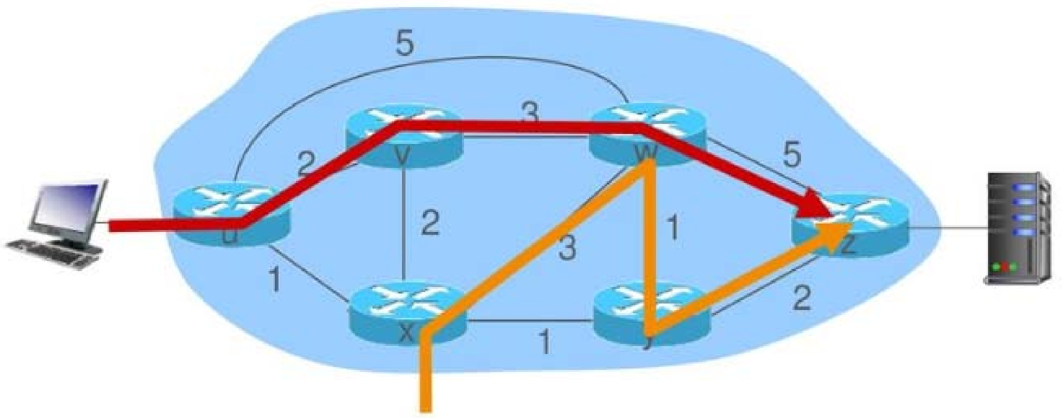

A
B
C
D
1
1+e
0
0
0
e
1
1
e
initially
2+e
0
1+e
1
0
0
given these costs, find new routing…. resulting in new costs
0
2+e
0
0
1
1+e
given these costs, find new routing…. resulting in new costs

How to achieve optimal routing dynamically?

Bugs: Traffic engineering is a hard problem!



Q: What if network operator wants to split u-to-z traffic along with uvwz and uxyz (load balancing)?
A: Cannot do it (or need a new routing algorithm)



Q: What if "w" wants to route yellow and red traffic differently?
A: Cannot do it (with destination based forwarding).

Historical attempts to address shortcomings

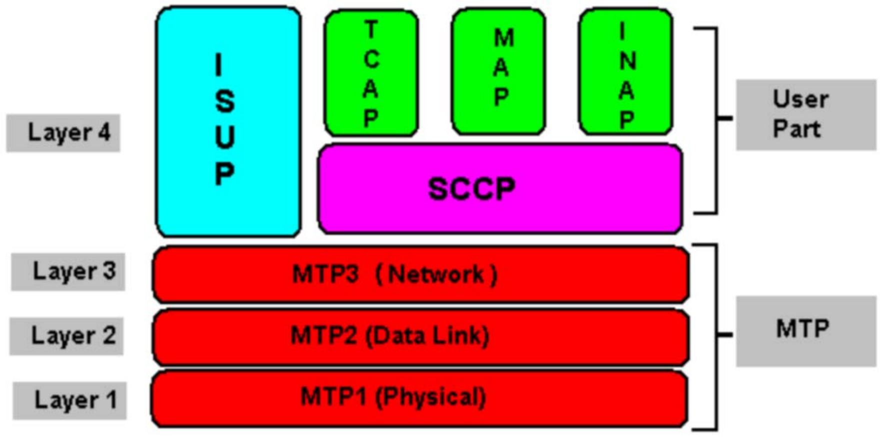
- Separation of control and data planes
  - Intelligent Networks (INs)
  - Common Open Policy Service (COPS)
  - Forwarding and Control Element Separation (ForCES)
- Network programmability
  - Active Networks
  - Programmable Switches
  - Network virtualization

Intelligent Network (IN)

A telephone network architecture that separates signaling (i.e., control) from switching (i.e., data)

- ❖ IN was developed in mid ‘80s by Bellcore (now Telcordia)
- ❖ Became industry standard (ITU-T Q.1200 series recommendations)

- ❖ Intelligence provided by network nodes on the service layer, distinct from the switching layer of the core network
  - ▪ IN nodes developed/programmed by telecommunications service providers (telephone company or mobile phone operator)
- ❖ SS7 is a main part of IN

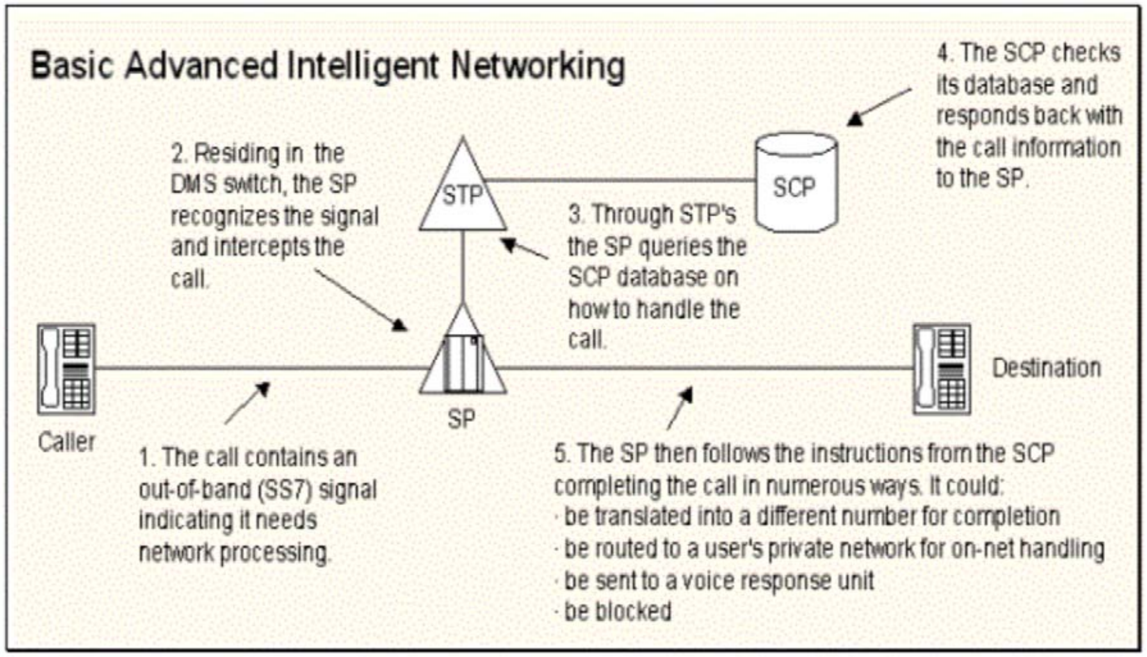


IN key components

- SP: Intercepts call and forwards it over a control network for further processing
- SCP: Contains service logic and tells SSPs how to handle a call
- SS7: Out-of-band signaling network
- Separation of SCP from SP allows addition of new features without modifying switching equipment

Examples of IN services

- Call routing based on location/time of call
- Toll-free calls/Freephone (800 numbers)
- Prepaid calling
- Virtual private networks
- Private-number plans
- Mass-calling service
- Premium Rate calls
- Call queuing
- Call transfer


Basic Advanced Intelligent Networking
4. The SCP checks its database and responds back with the call information to the SP.
2. Residing in the DMS switch, the SP recognizes the signal and intercepts the call.
STP
SCP
3. Through STP's the SP queries the SCP database on how to handle the call.
Caller
SP
Destination
1. The call contains an out-of-band (SS7) signal indicating it needs network processing.
5. The SP then follows the instructions from the SCP completing the call in numerous ways. It could:
· be translated into a different number for completion
· be routed to a user's private network for on-net handling
· be sent to a voice response unit
· be blocked

Policy Based Networking (PBN) and its Policies

- Client/server model for flexible support of QoS, accounting and security policies
- Provides operators with a central point to describe policies
- Provides mechanisms for devices to obtain applicable policy from a central server instead of requiring manual configuration
- Common Open Policy Service (COPS) standardized by IETF early 2000s
  - RFC 2748

Example of policy:

Give administrators high priority when accessing the network resources

Policies are Abstract, Goal oriented

"WHAT" instead of "HOW"

E.g.: Administrators (10.1.1.x) must have high priority (DSCP=6)

HOW approach: Mark 10.1.1.x traffic with DSCP=6

WHAT approach: Give high priority to Administrators

The policy is still valid even if:

Topology and/or service implementation changes

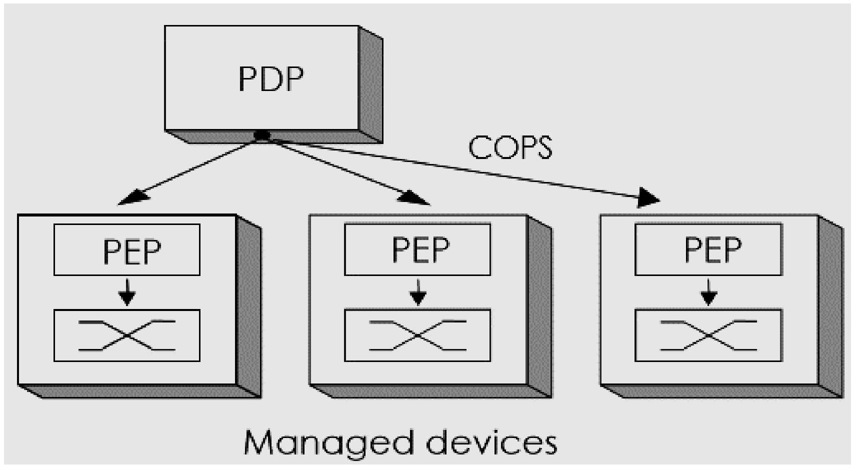
Administrators subnet is changed/expanded

Administrators do not need to be associated with a specific subnet

Common Open Policy Service (COPS)

Policy Enforcement Point (PEP)

- Network elements where policies are applied (e.g., DIFFSERV router)
- Can perform admission control, packet classification, bandwidth metering, etc.



Policy Decision Point (PDP)

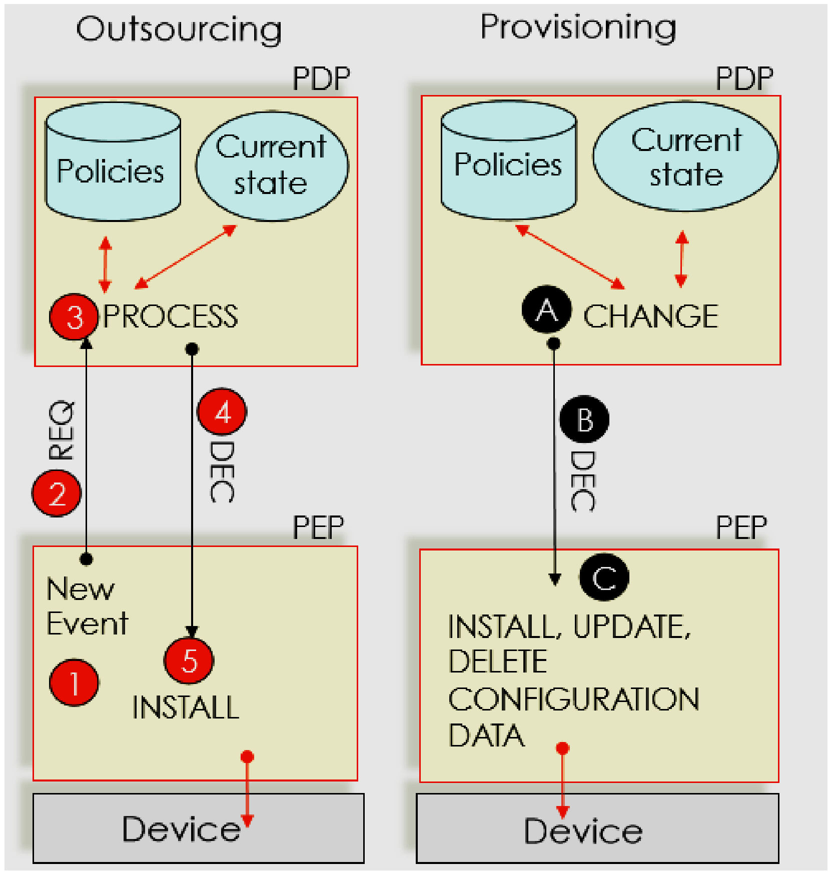
- Central server that provides PEPs with policy information (e.g., VoIP traffic should be prioritized now)

COPS Protocol

- Facilitates communication between PEPs and PDP

COPS Protocol

- COPS client-types
  - Each PEP implements one or more clients of various client-types
  - Client-types are defined in separate documents (standard or vendor specific)
- COPS-RSVP and COPSPR are such clients
  - COPS Operation
  - Outsourcing
  - Provisioning



ForCES

- FORwarding and Control Element Separation (ForCES)
- Separates control and forwarding of IP network
  - Control Element (CE) implements routing algorithms, admission control, signaling
  - Forwarding Element (FE) performs packet forwarding, queuing, traffic shaping, etc.
  - Each FE is associated with a CE
- Provides open and standardized API to enable rapid innovation in both the control and forwarding planes.
- Standardized by IETF early 2000s
  - April 2004: ForCES Framework (RFC3746)
  - March 2010: FORCES protocol (RFC5810)

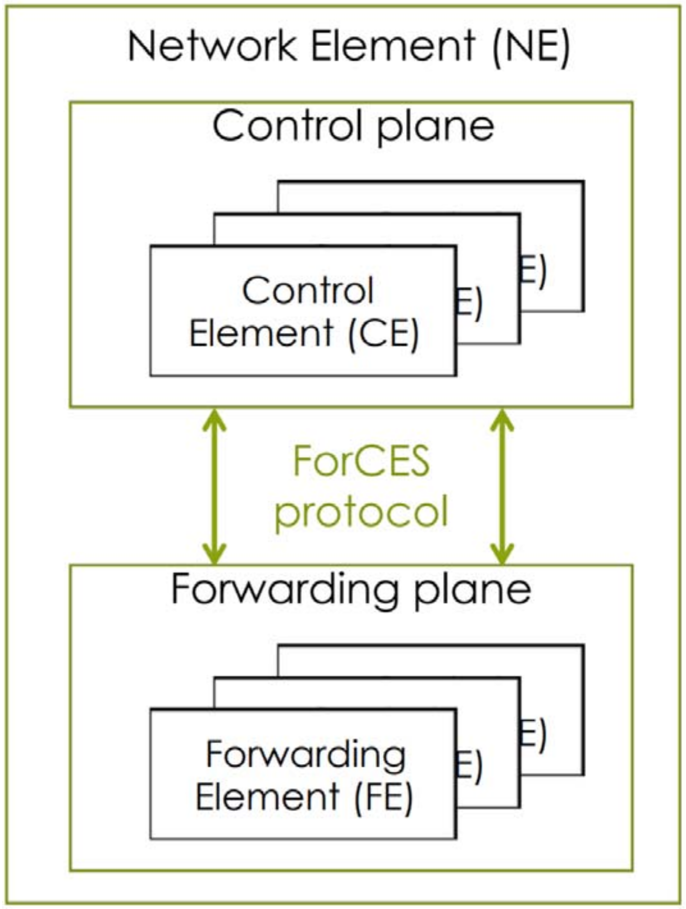
ForCES Framework and Protocol

Network Element (NE)

- Packet Processing Entity
- Constitutes of CEs and FEs
- CEs/FEs Physical or Virtual

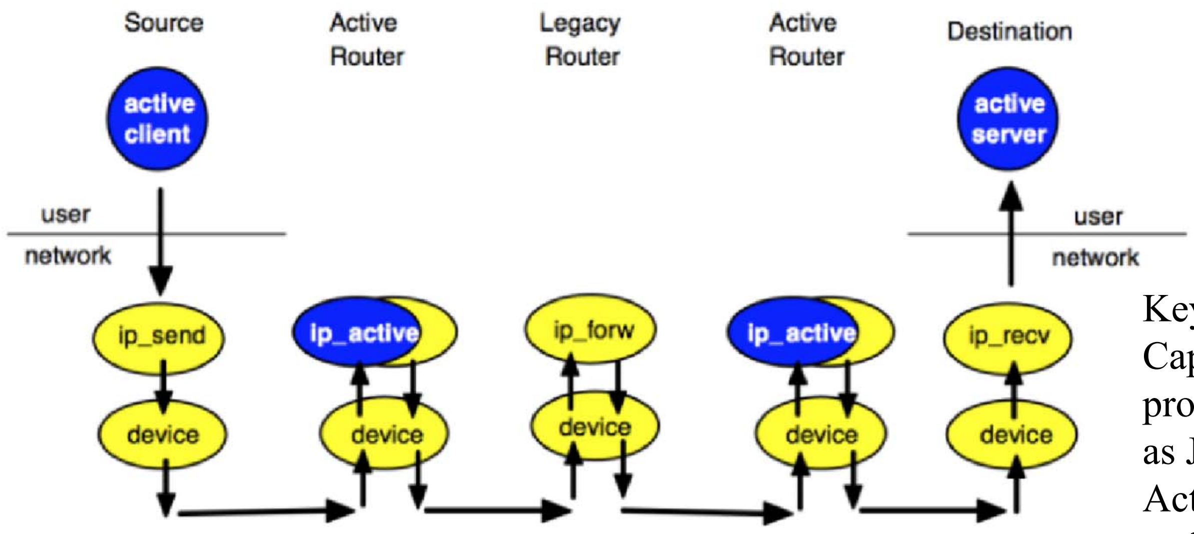
ForCES protocol

- Simple Commands (Set/Get)
- Message Ack (Always/Never/On Failure/On success)
- Transactional capability (2 Phase Commit)
- Various Execution modes (Execute all or none, till failure, on failure)
- Security (IPSec)



Active Networks

❖ Motivated by "Internet Ossification" (DARPA research community 1995)
  - Difficulty of integrating new technology
  - Difficulty accommodating new services
  - Key idea is to embed programs into packets and execute them on "active nodes"
  - Active nodes can co-exist with legacy devices



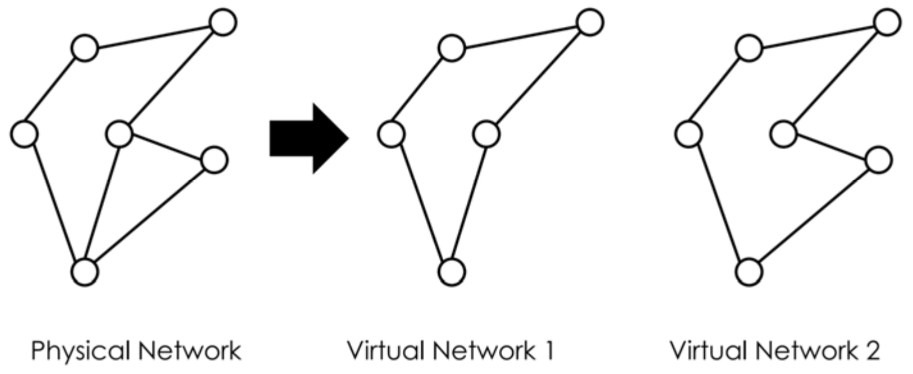
Key concepts:
Capsules: Packets carrying programs or fragments of a program (e.g., MIT ANTS project implemented Capsules as Java programs)
Active nodes: Network devices capable of interpreting and executing programs in a capsule

Programmable Switches

- ❖ Executable code for packet processing injected into devices (out of band) rather than into packets
  - o Out-of-band switch programing
  - o Operator-driven switch extensibility
- ❖ Packets are routed through such programmable switches for custom processing
- ❖ Example projects:
  - o OPENSIG (Columbia): provide access to network hardware via open, programmable network interfaces
    - • Allows the deployment of new services through a distributed programming environment (e.g., xBind)
  - o SwitchWare (Upenn): addressed security concern using formal verification techniques to ensure correct behavior of programs installed on programmable switches

Network Virtualization

- Another response to so-called Internet ossification
  - Lots of band-aids and makeshift solutions (e.g., overlays)
- A new architecture (aka clean-slate) is needed
  - Hard to come up with a one-size-fits-all architecture
  - Almost impossible to predict what the future might unleash
- Why not create an all-sizes-fit-into-one architecture instead
  - Open and expandable
  - Coexistence of heterogeneous architectures
  - (at least) Testbed for future networking architectures and protocols



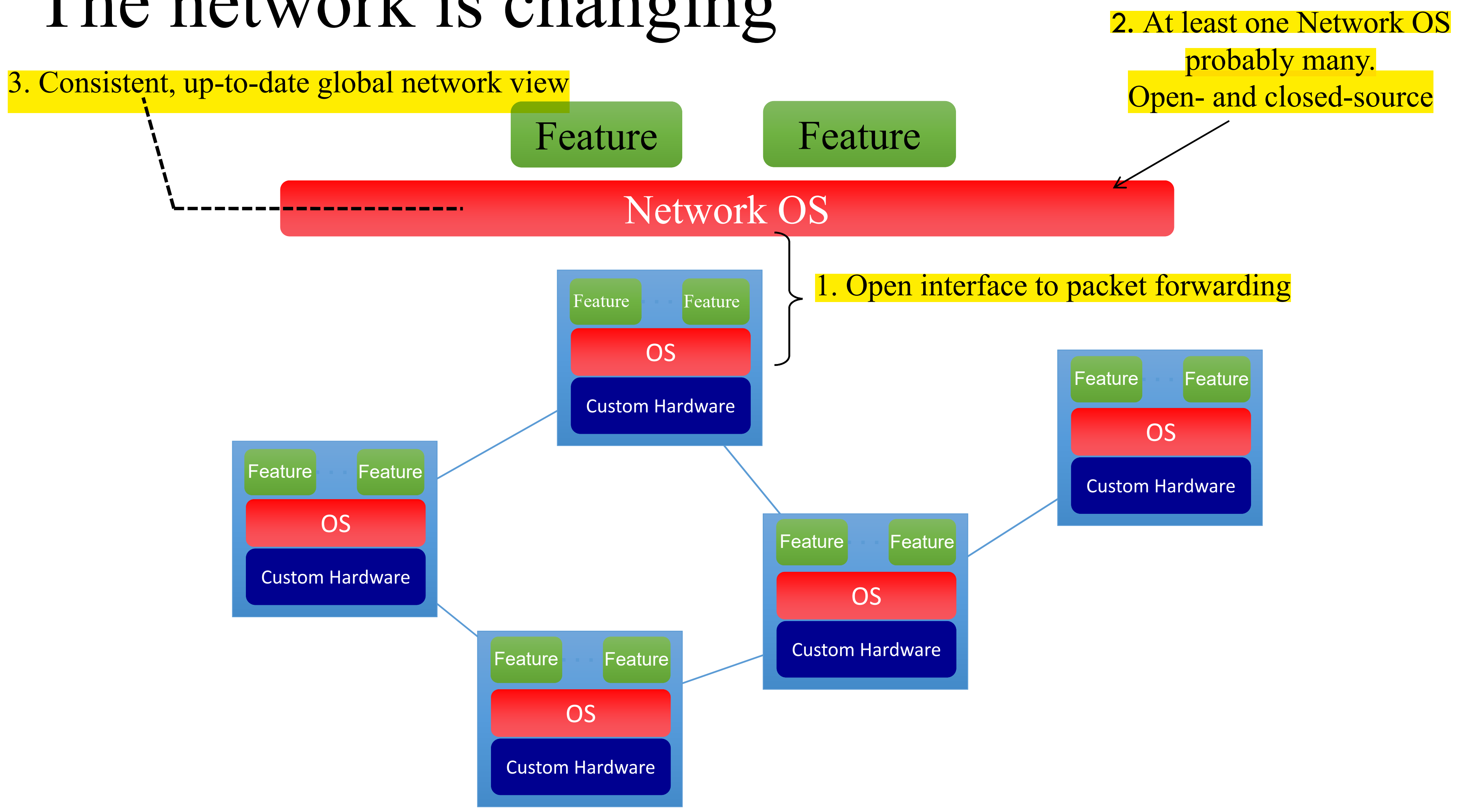
Lessons Learned from Previous Attempts

- Separation of control and data planes also separates the development cycles of new features (software) from hardware
- Accelerating time to market for new services/features
  - Many past attempts to decouple control and data were not deployed in practice because of their need for new hardware
- Requirement of deploying new hardware or hardware redesign is also a big barrier to entry
  - Requirement for standardization is another big barrier to entry for a new architecture/protocol
- Timing is key
  - Without a clear use-case technology adoption can fail
- Backward compatibility

Requirements for Future Networks

- Programmability
  - Network's behavior can be changed dynamically and quickly
- Automation
  - Apply policy change without requiring manual reconfiguration of devices
- On-demand Scaling
  - Scale out/in resources for network services based on demand
- Mobility
  - Handle mobility of network end-points and services
- Extensibility
  - Introduce new features with minimal disruption of operations
- Scalability
  - Huge amount of new devices with diverse quality of services
- Independent
  - Development of Hardware & Software
- Accelerate
  - Innovation by separating the software from the hardware

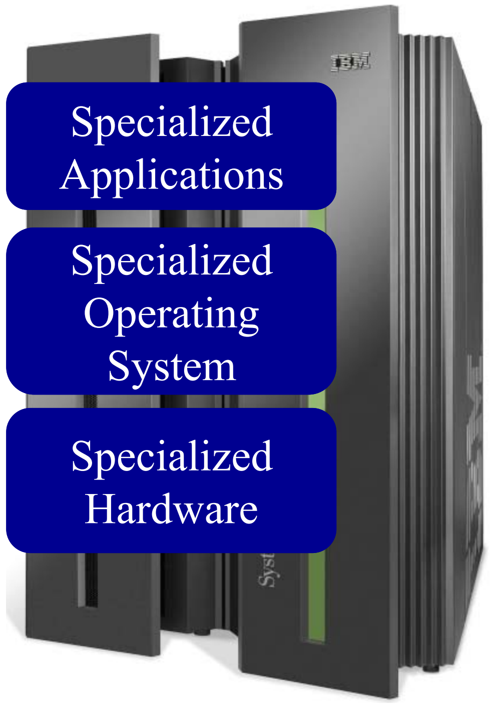
The network is changing

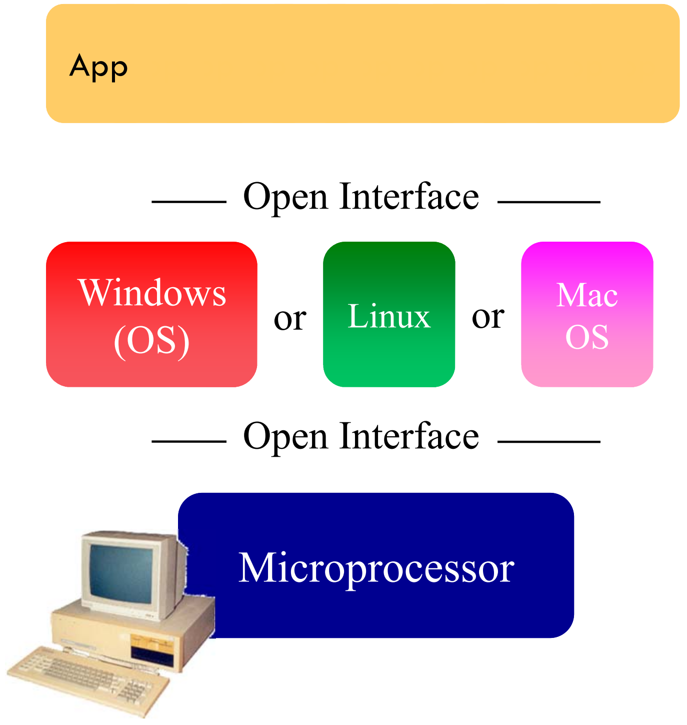

3. Consistent, up-to-date global network view
2. At least one Network OS
probably many.
Open- and closed-source
Feature
Feature
Network OS
1. Open interface to packet forwarding
Feature
Feature
OS
Custom Hardware

SDN Advent

- ~2005: renewed interest in rethinking network control plane
- Software Defined Networking (SDN) emerged as a new networking architecture that
  - Separates control plane from data plane
  - Software running on a logically centralized server remotely controls network hardware
  - Open standards (**vendor neutral**)
  - Directly programmable
  - Facilitates automation & programmability

Analogy: OS Evolution

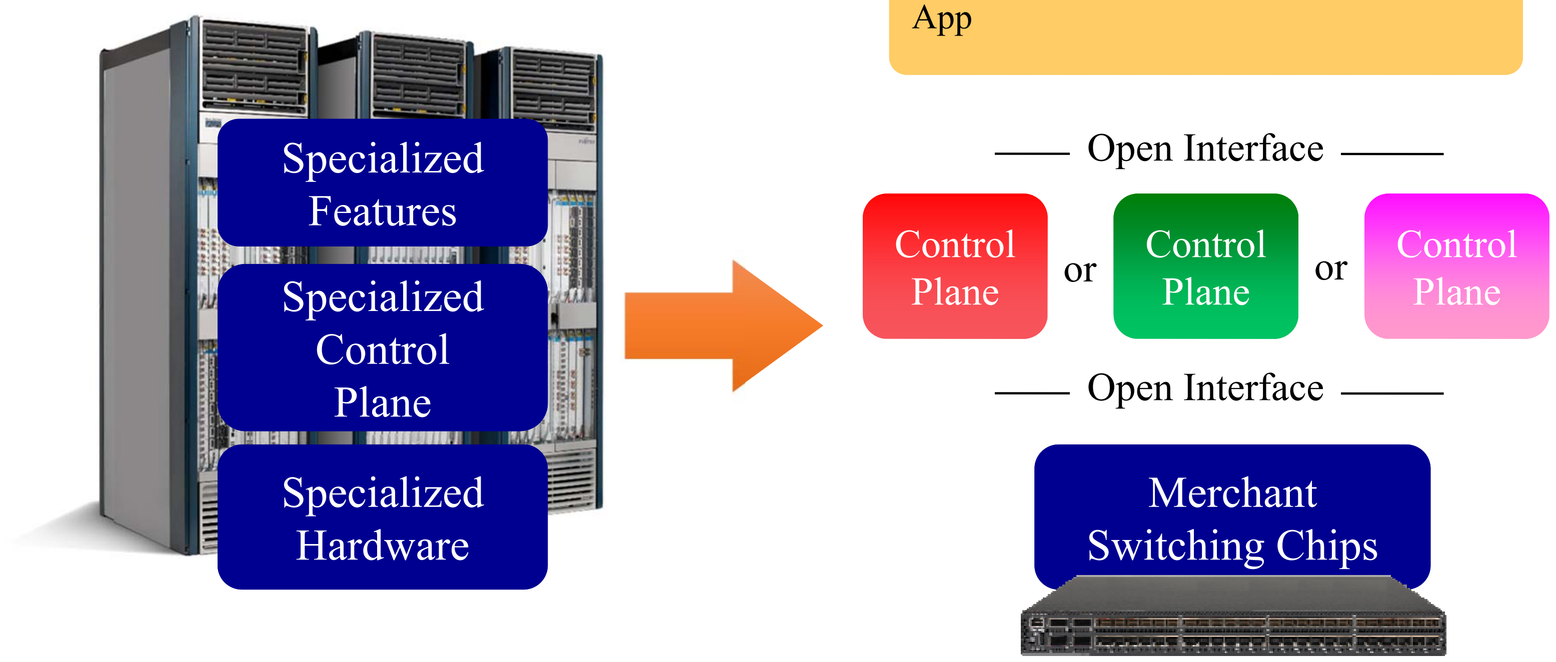




Vertically integrated
Closed, proprietary
Slow innovation
Small industry

Horizontal
Open interfaces
Rapid innovation
Huge industry

Similar network evolution with SDN



Vertically integrated
Closed, proprietary
Slow innovation

Horizontal
Open interfaces
Rapid innovation

SDN Advantageous

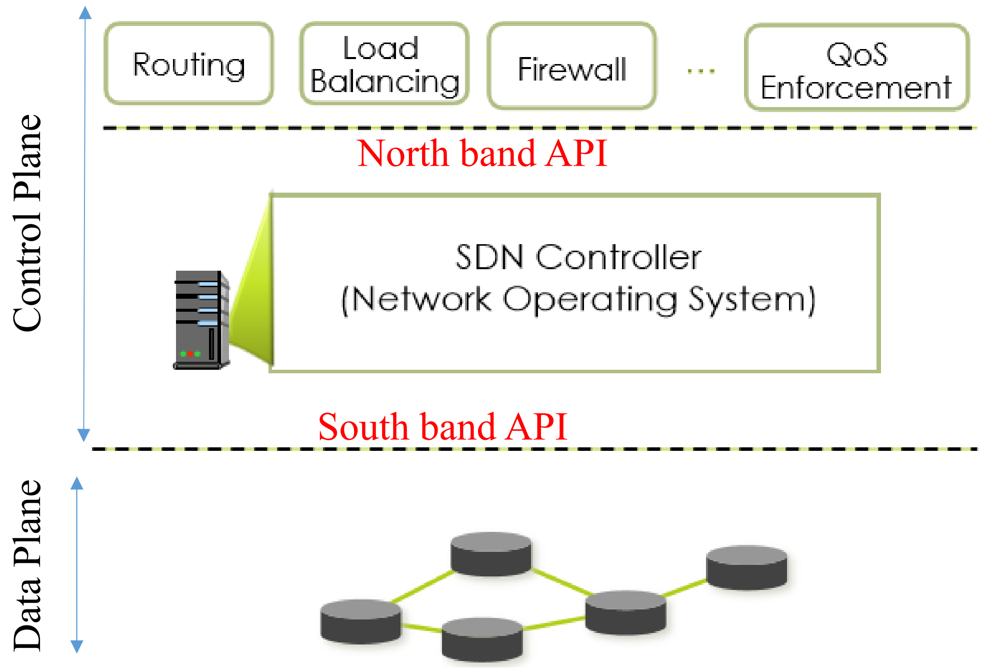
- Separation of control and data planes
  - Enables independent development of new features without hardware modification
- Global network view at control plane
  - Network-wide view allows easier application of policies
- Management Automation
  - Minimize human intervention
  - Lower operational cost
- Monitoring
  - On-demand monitoring of resources
  - Dynamically change what to monitor
  - Application performance can be improved by advanced monitoring features

SDN Advantageous

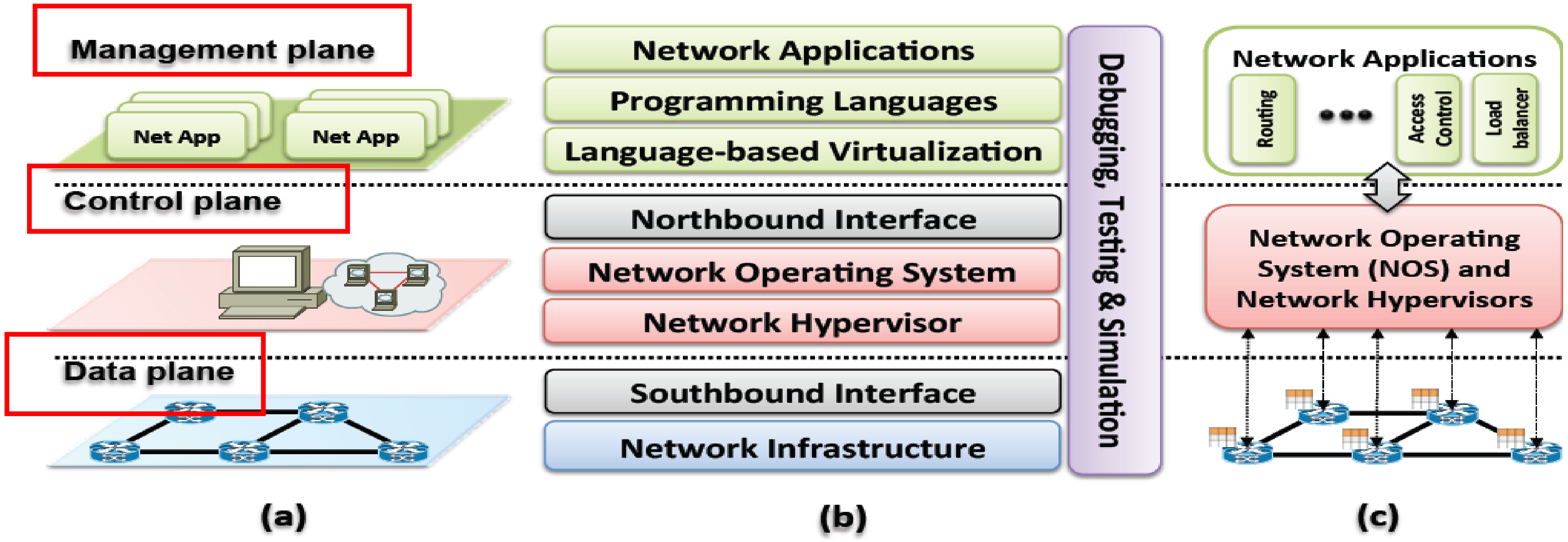
- Elasticity and Scalability
  - Resources can be scaled up or down on-demand
- More Flexible/Easier Performance Management
  - Traffic engineering
  - Capacity optimization
  - Load balancing
  - Faster failure recovery
- Network Function Integration
  - On-demand provisioning of network functions
    - Firewalls, IDS, IPS, etc.

Components of SDN

- Data plane switches
  - SDN controlled switches
- South bound API
  - Interface between control and data plane (e.g. OpenFlow)
- SDN controller
- North bound API
  - Interface between controller and control apps



SDN Architecture and its Fundamental Abstractions



Software-Defined Networks in (a) planes, (b) layers, and (c) system design architecture

Communication layer

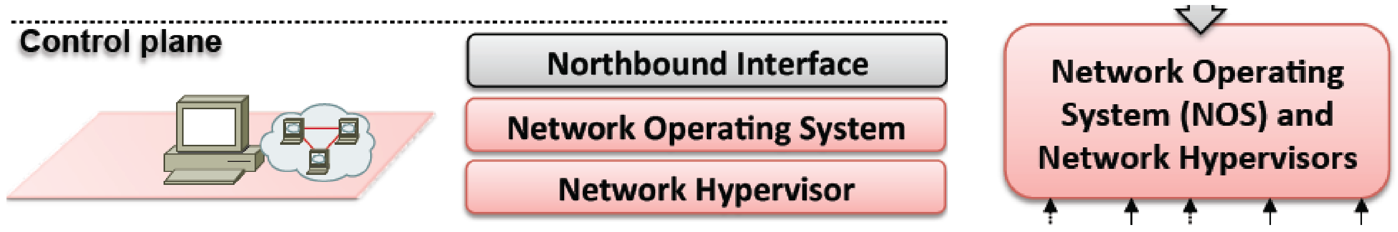
- Implements protocols (e.g., OpenFlow, OnePK etc.) to transfer information between the controller and network devices
- Provides controller a uniform way to communicate with devices from different vendors
- Provides mechanisms for receiving asynchronous event notifications (e.g., link up/down) from network devices.

Layer 3: Management Plane

is the set of applications such as routing, firewalls, load balancers, monitoring, and so forth. Essentially, a management application defines the policies, which are ultimately translated to southbound-specific instructions that program the behavior of the forwarding devices.

Management Plane (MP)

- Network applications can be seen as the “network brains
- Major functionality:
  - Traffic engineering
  - Mobility & wireless
  - Measurement & monitoring
  - Security & Dependability
  - access control
  - QoS management
- API provided by SDN controller
- Unbundled: can be provided by 3rd party: distinct from routing vendor, or SDN controller

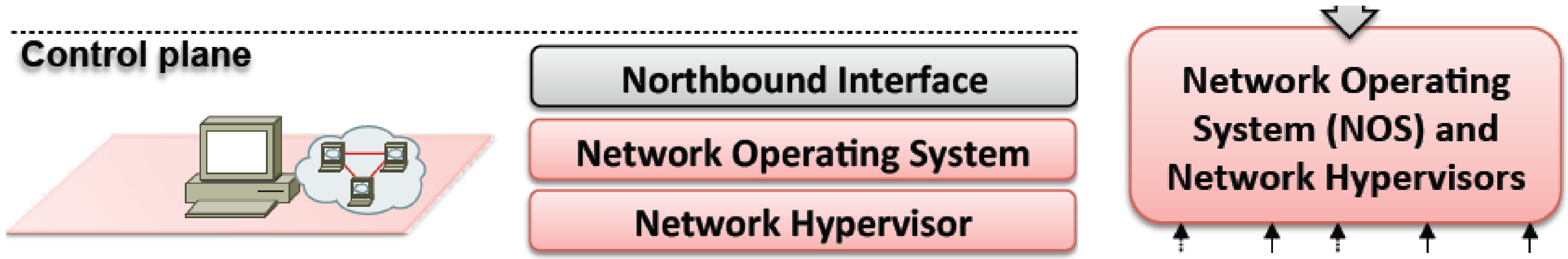


Control Plane

Forwarding devices are programmed by control plane elements through well-defined SI embodiments. The control plane can therefore be seen as the “network brain”. All control logic rests in the applications and controllers, which form the control plane.

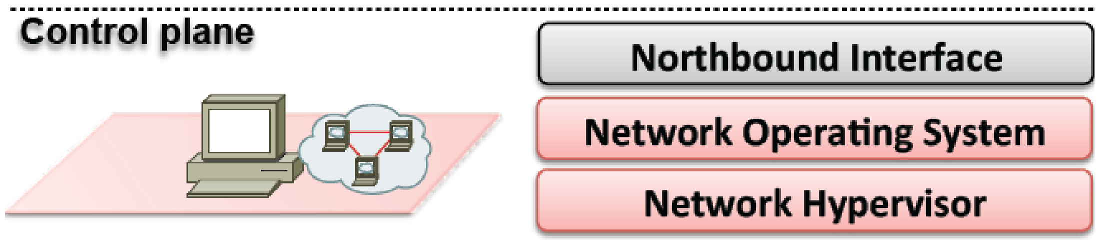
Control Plane Functions

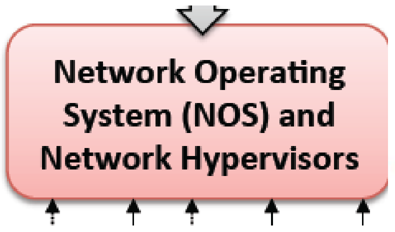
- Topology Management
  - Runs topology discovery protocol to discover network devices and their interconnections
- Path computation
  - Uses topology information to compute shortest path and setup necessary forwarding rules for routing flows.
- Notification Management
  - Receives, processes and forwards network event information to applications
- Statistics Collection
  - Collects statistics from network devices by reading different counters
- Device Manager
  - Configures network devices



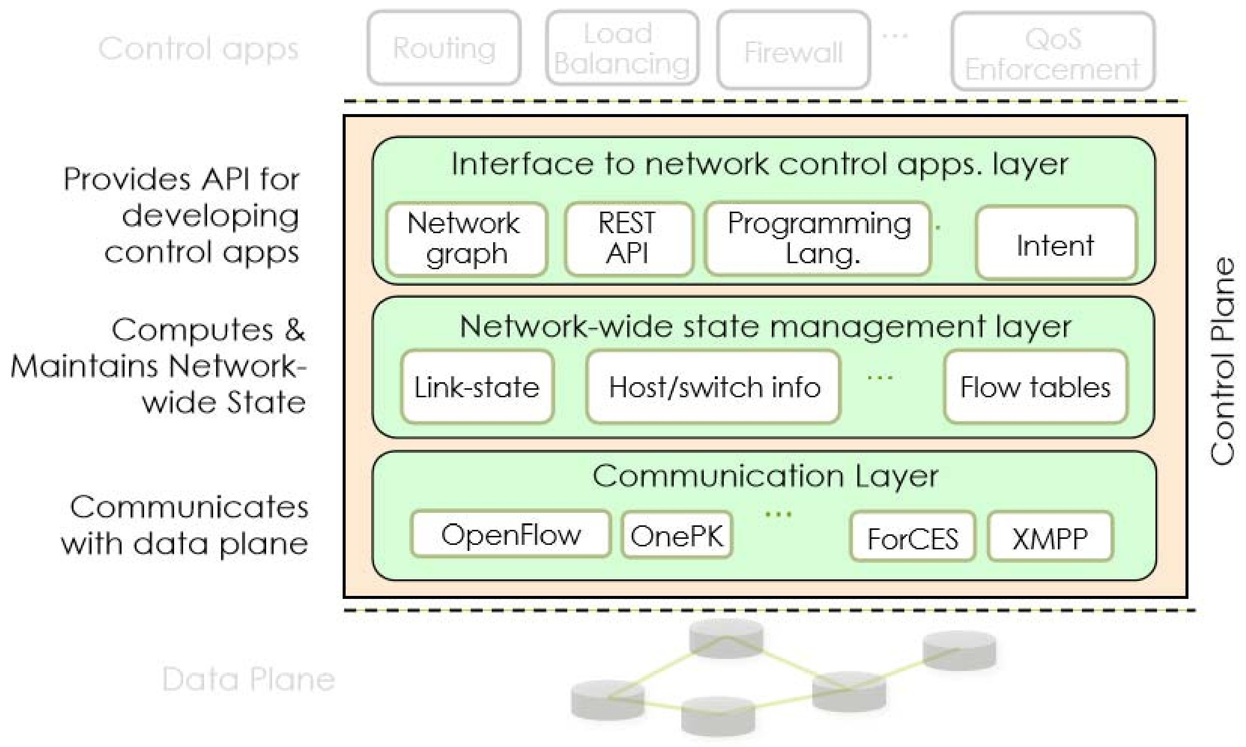
Control Plane Elements

- **Network Operating Systems (NOS):**
  - NOS (or controller) is a critical element in an SDN architecture as it is the key supporting piece for the control logic (applications) to generate the network configuration based on the policies defined by the network operator
  - Logically centralized software to control data plane switches
  - Maintain network state information
  - Interacts with network control applications “above” via northbound API
  - Interacts with network switches “below” via southbound APIs
  - Supports one or more SB-APIs
- **Controllers**
  - OpenFlow: NOX/POX, Floodlight, OpenDaylight, ONOS, etc.
  - Other: Path Computation Element (PCE) in MPLS/GMPLS, ForCES, etc.



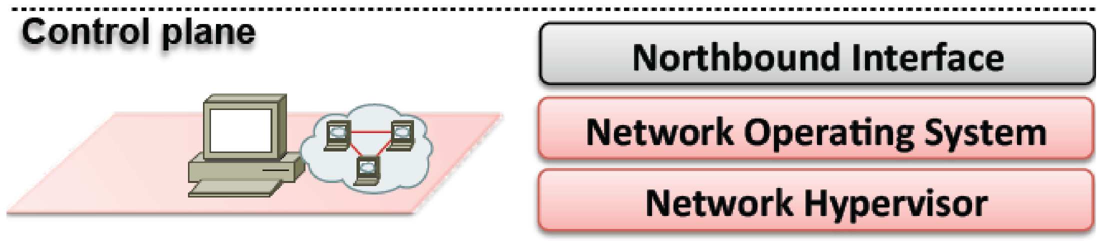


Components of SDN Controller



Northbound Interface (NI)
Interface to control apps layer

- ❖ The network operating system can offer an API to application developers. This API represents a northbound interface, i.e., a common interface for developing applications. Typically, a northbound interface abstracts the low level instruction sets used by southbound interfaces to program forwarding devices
  - o Used by control apps to communicate with the controller
  - o obtain network state
    - flow table content, counters, device health, etc.
  - o modify network state
    - add/remove flow table entry, turn on/off a switch port, etc.
  - o Used by controller to provide network-wide view to control apps
- ❖ Northbound API exists in many forms
  - ▪ Programming Language
  - ▪ Intent Language
  - ▪ RESTful API/Low level sockets programming



Intent

Intent: Policy based derivatives describing *“what”* needs to be performed rather than *“how”* it needs to be performed

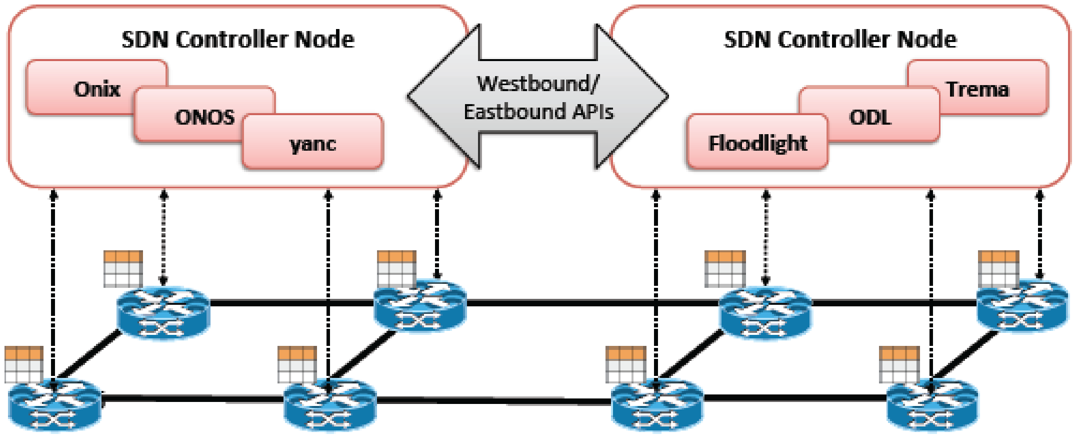
What: Ensure connectivity between switch A and switch B

How:
- Setup flow between switch A and B
- Continuously monitor all switches on the path
- If something fails re-compute and reinstall flow rules

- ❖ Intent abstracts low level complexity from control applications
- ❖ Control applications specify their desired behavior through an intent framework supported by the controller
- ❖ Controller compiles intents to low level operations (e.g., flow installation) and takes the necessary actions

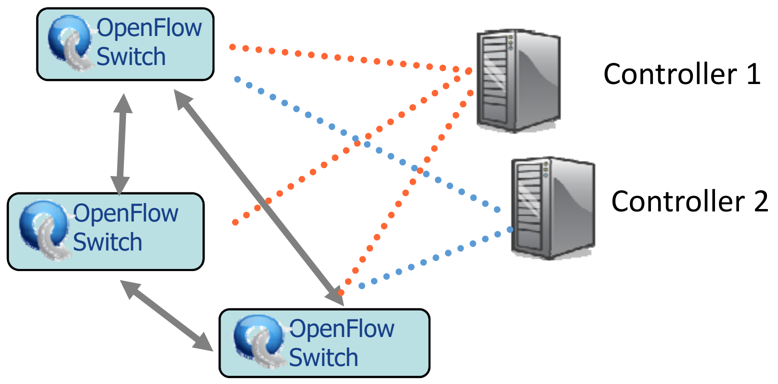
Eastbound and Westbound

- ❖ In the multiple controllers setup
- ❖ In the multi-domain setup,
- ❖ In heterogonous setup



Distributed controllers: east/westbound APIs.

Multiple Controllers scenario is possible



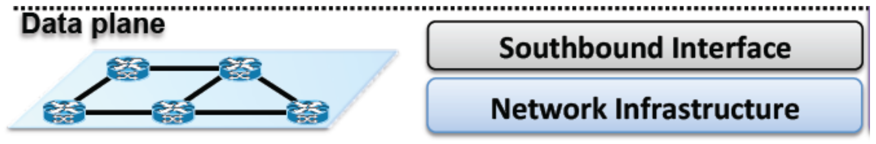
Data Plane

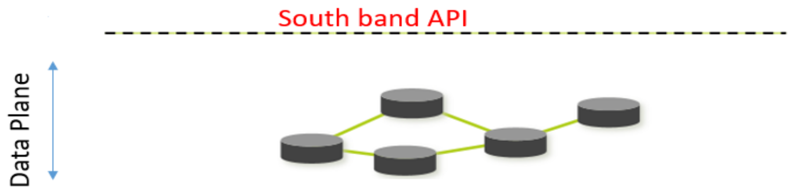
Forwarding devices are interconnected through wireless radio channels or wired cables. The network infrastructure comprises the interconnected forwarding devices, which represent the data plane.

Layer I

- ❖ Layer 1 includes network infrastructure and southband interface.
  - o An SDN infrastructure, similarly to a traditional network, is composed of a set of networking equipment (switches, routers and middlebox appliances). The main difference resides in the fact that those traditional physical devices are now simple forwarding elements without embedded control or software to take autonomous decisions.
- ▪ Forwarding Devices (FD) (Data Plane Switches):

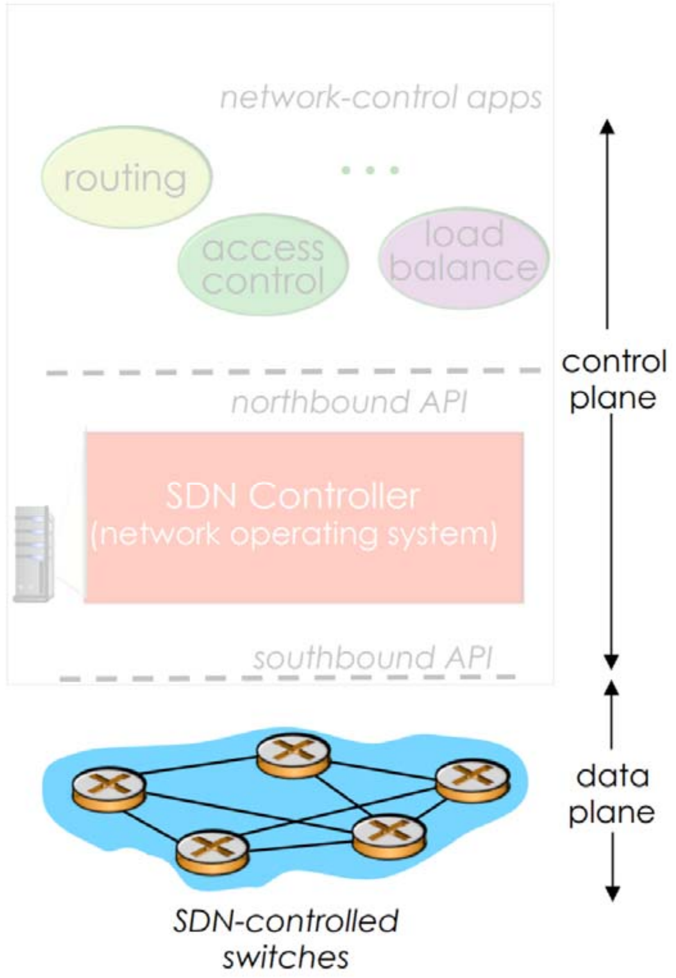
  Hardware- or software-based data plane devices that perform a set of elementary operations. The forwarding devices have well-defined instruction sets (e.g., flow rules) used to take actions on the incoming packets (e.g., forward to specific ports, drop, forward to the controller, rewrite some header).





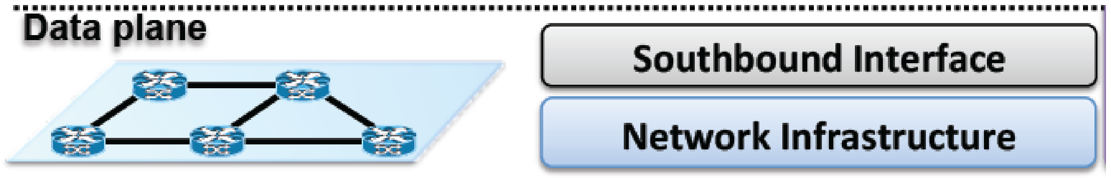
Data Plane Switches

- Programmable Network Devices
  - fast, simple, commodity switches implementing data-plane forwarding in hardware
  - can be software (virtual switch)
  - typically has no control functionality built-in
  - programmed by a remote controller
  - switch flow table computed, installed by controller
  - Supports different SB-APIs (e.g., OpenFlow)
  - Available from different vendors
    - Pica8, Pronto, HP, NEC, etc



South-band Interface (SI)

- The instruction set of the forwarding devices is defined by the southbound API, which is part of the southbound interface. Furthermore, the SI also defines the communication protocol between forwarding devices and control plane elements. This protocol formalizes the way the control and data plane elements interact.
- The SB-API is used by the controller to communicate with the data plane switches
  - defines what is controllable and what is not
- SB-API can be for
  - Control (e.g., OpenFlow)
  - Configuration (e.g., OFCONFIG)
  - OpenFlow is the most popular SB-API for SDN
    - Implemented over TCP
    - Supports both in-band and out of-band control



South-band Interface (SI)

- Southbound API supports communication between SDN control and data plane
  - TCP is used to exchange OpenFlow messages between a switch and the controller
- Port 6633 and 6653 have been reserved by IANA
  - TLS encryption is supported (optional)
- Three classes of OpenFlow messages:
  - Asynchronous switch-to-controller
  - Controller-to-switch
  - Symmetric messages

Examples of N/S I

Example NI: ONOS Intent Framework

- Allows control applications to express intent by specifying different constraints on network resources
- Has two major components
    - Intent Compiler: Translates intents to installable actions on the network
    - Intent Coordinator: Determines how the network will be programmed/monitored to implement the intent

Example NI: Programming Languages

- Higher level languages allowing control applications to
  - Read/monitor network states
    - What is in the meter table?
  - Compute policies
    - What should be the new meter table entry?
  - Write network state
    - Modify the meter table to reflect new policy
- In contrast to Intents, programming languages can be used to specify both “what” and “how”

Example NI: Programming Languages (cont)

- Control programs are written in higher level abstractions. A compiler translates them to OpenFlow actions
  - Example: Pyretic, Kinetic ! Pyretic
- Language primitives closely resemble the match+action abstraction of OpenFlow. e.g., match(src="…") | match(dst="..") >> fwd(1)
  - Higher level primitives are translated to OpenFlow messages by the compiler
- Kinetic
  - An event-based programming language for SDN Can be used to express policies for different application-defined event, e.g., if user data connection goes beyond a bandwidth cap in a given month, rate limit user data connection

Example SI -API : OpenFlow

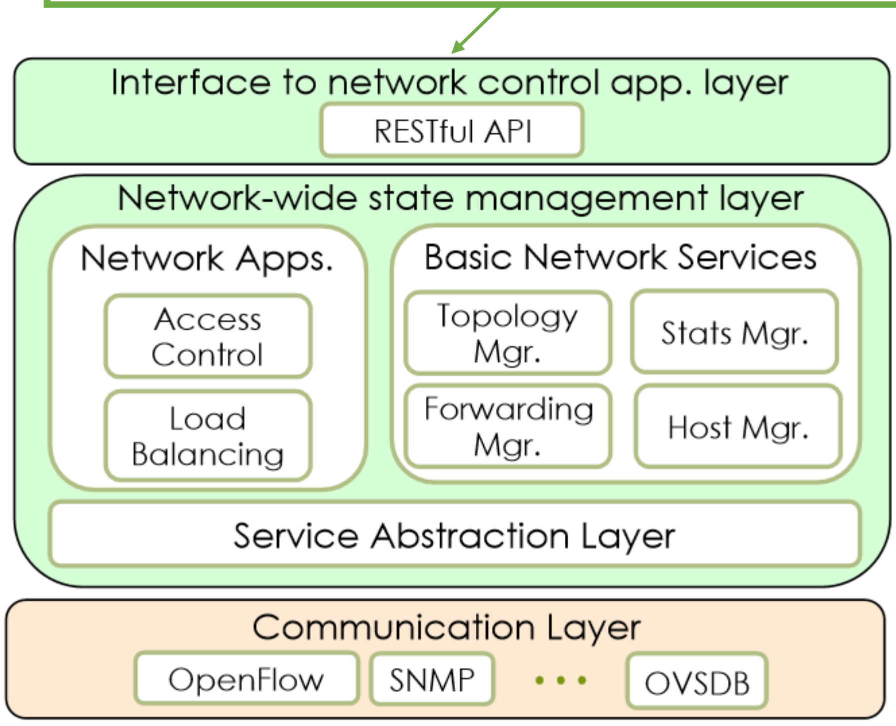
- Southbound API supports communication between SDN control and data plane
- TCP is used to exchange OpenFlow messages between a switch and the controller
  - Port 6633 and 6653 have been reserved by IANA
- TLS encryption is supported (optional)
  - three classes of OpenFlow messages:
    - Asynchronous switch-to-controller
    - Controller-to-switch
    - Symmetric messages

Examples of SDN Controllers

OpenDaylight Controller

- ❖ An open source Java based SDN controller started as a Linux foundation project in 2013 and the first code release was in 2014
- ❖ Currently deployed by major telecom operators and also in enterprise networks
- ❖ Designed as a single standalone controller which also supports cluster mode operation.
- ❖ Communication layer supports a wide range of south-bound APIs contributed by different vendors
  - OpenFlow
  - Cisco OnePK
  - XMPP
  - SNMP
  - OVSDB
- ❖ A set of basic network services comes shipped with the controller
- ❖ Service Abstraction Layer
  - provides an uniform way to communicate with the network
  - hides the heterogeneity in communication layer

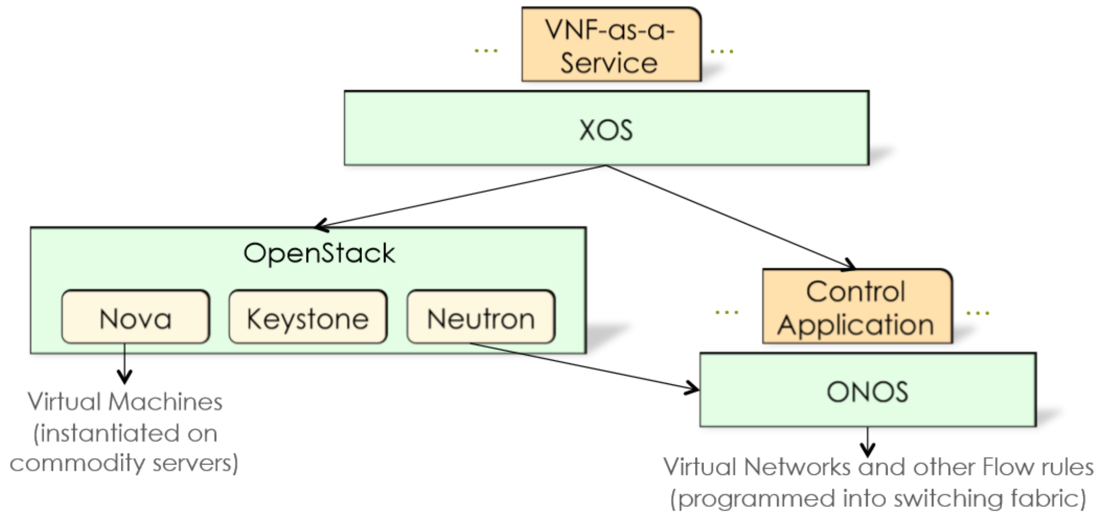
- RESTful API exposed for control apps.
- Network apps may be integrated with, or be external to OpenDaylight ! External apps use RESTful API
- Integrated apps use APIs of Service Abstraction Layer and require recompiling the controller



ONOS Controller

- ❖ Open source Java based SDN controller initiated by ON.Lab in 2014 and later became a Linux foundation project in 2015

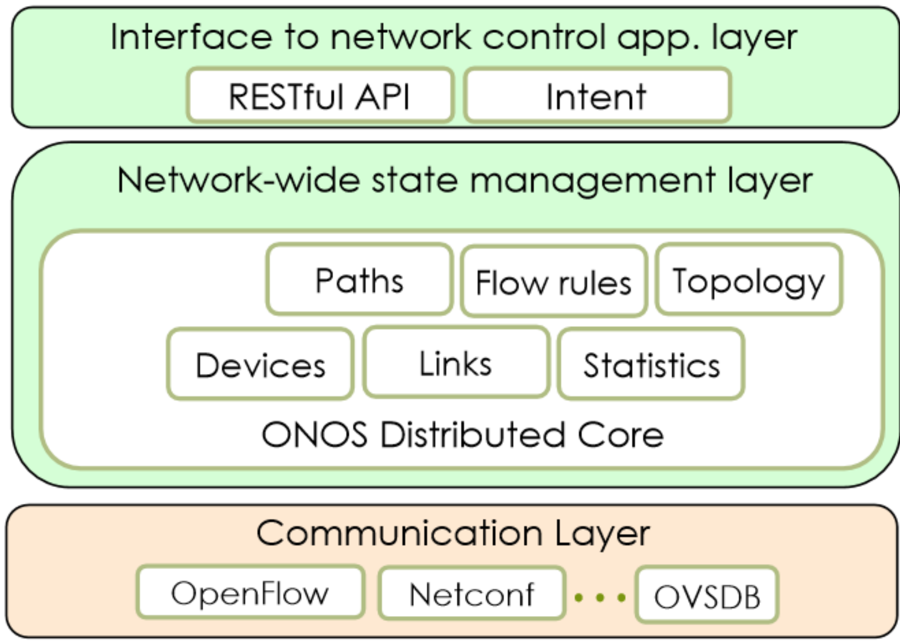
CORD – Software Stack



VNF-as-a-Service
XOS
OpenStack
Nova
Keystone
Neutron
Control Application
ONOS
Virtual Machines (instantiated on commodity servers)
Virtual Networks and other Flow rules (programmed into switching fabric)

ONOS Controller

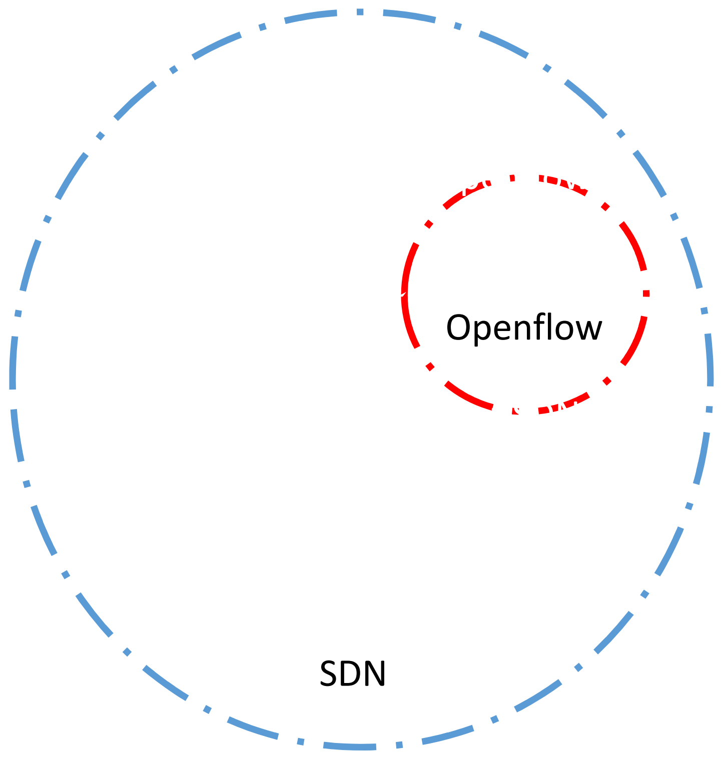
- Designed as a distributed controller
  - Considerable emphasis on service reliability, replication performance scaling
- Communication layer
  - mostly supports OpenFlow
  - limited support for other southbound APIs
- ONOS ships with some basic network services
- Control apps external to controller
- Intent framework
  - high-level specification of service
  - what rather than how
- Distributed Core
  - service reliability ! replication ! performance scaling



OpenFlow

OpenFlow

- OpenFlow does not equal SDN
- OpenFlow is one flavor, or a subset, of SDN
- OpenFlow is a Layer 2 communications protocol that gives access to the forwarding plane of a network switch or router over the network



OpenFlow Data plane Architecture

- OpenFlow specifications define a common logical architecture for OpenFlow data planes
- Main elements :
  - Control channel for communicating with controller
  - Ports for sending/receiving packets
  - Flow table to decide actions on packets based on packet headers
  - Group tables to abstract operation on groups of ports
  - Meter table for QoS enforcement

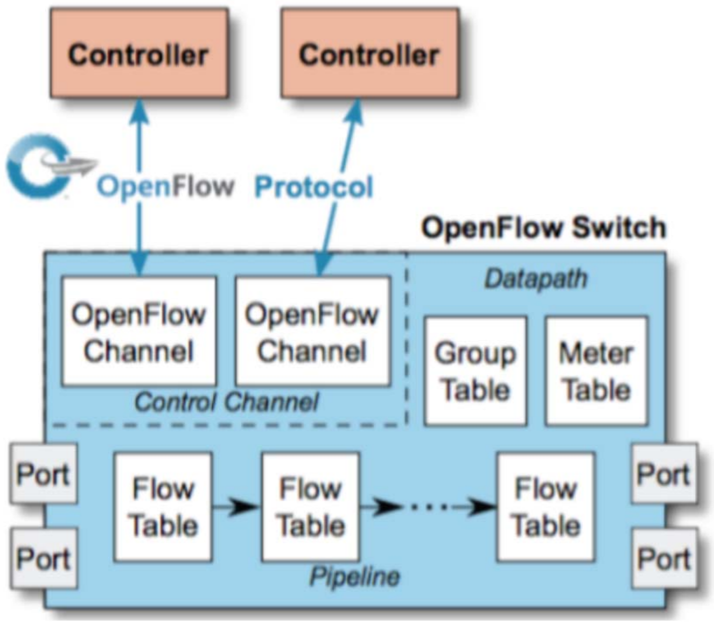


Image source: https://wiki.onosproject.org/display/ONOS/OpenFlow+1.5+Implementation

Control Channel

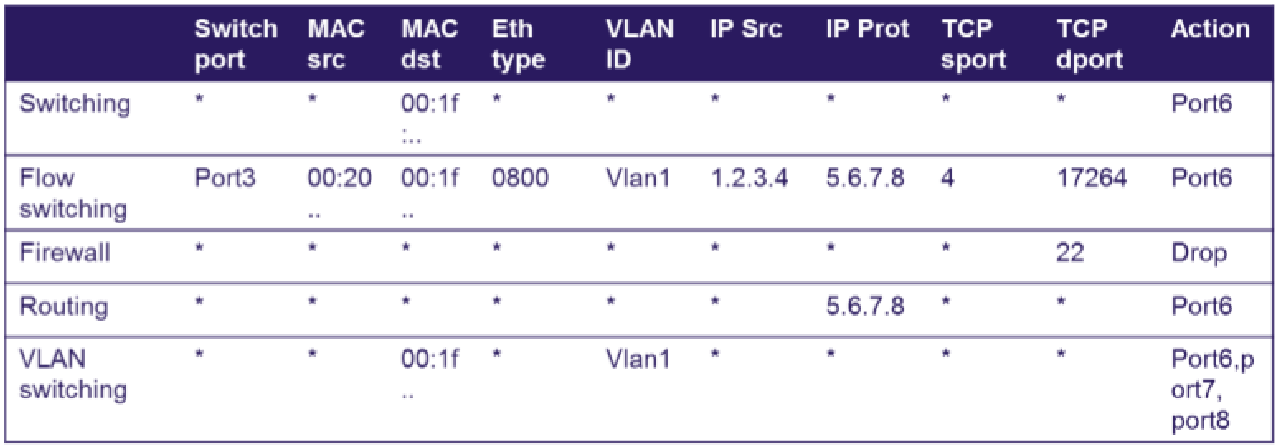
- Each OpenFlow switch connects to the SDN controller over a TCP connection
- OpenFlow protocol defines a set of messages between the switch and the controller that allow:
    - Controller to program flow tables on the switches
    - Controller to query information from switches
    - Switches to send asynchronous notifications to the controller
  - Control channel can be
    - In-band: Switches have default rules to forward control packets
    - Out-of-band: Control network can be a non-SDN network

Ports

- Ports are abstractions for switch interfaces where packets can be pushed to/pulled from
- Three types of ports:
  - *Physical*: corresponds to a hardware interface
  - *Logical*: higher level abstractions that might not physically exist, *e.g.*, representing multiple physical ports as a single logical port
  - *Reserved*: restricted-use ports as defined by OpenFlow specification, *e.g.*, port used for communication with controller

Flow Table

- ❖ Flow table matches incoming packets against a set of forwarding rules (each rule is called a flow table entry)
- ❖ For faster packet matching, the flow table is implemented in hardware using Ternary Content Addressable Memory (TCAM)
  - ▪ expensive and power-hungry thus very limited resource (there is a chance for congestion)
  - ▪ can hold limited flow table entries, therefore, old entries are evicted



	Switch port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Prot	TCP sport	TCP dport	Action
Switching	*	*	00:1f :..	*	*	*	*	*	*	Port6
Flow switching	Port3	00:20 ..	00:1f ..	0800	Vlan1	1.2.3.4	5.6.7.8	4	17264	Port6
Firewall	*	*	*	*	*	*	*	*	22	Drop
Routing	*	*	*	*	*	*	5.6.7.8	*	*	Port6
VLAN switching	*	*	00:1f ..	*	Vlan1	*	*	*	*	Port6,p ort7, port8

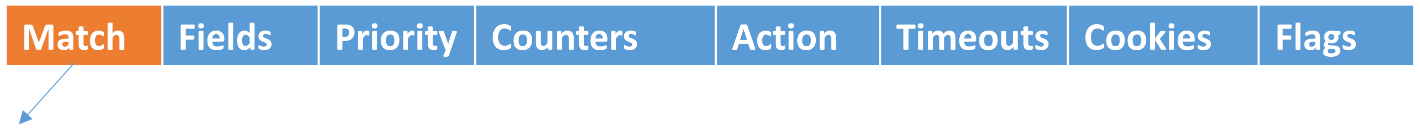
Flow Table Structure

- Flow table consists of rows called Flow table entries. An entry is structured as follows:

Match	Fields	Priority	Counters	Action	Timeouts	Cookies	Flags

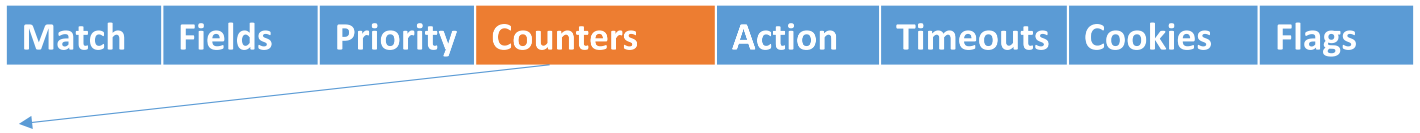
- Match fields: used to match packets to flows based on header fields
- Priority: breaks tie between conflicting matches
- Counters: per-flow entry statistics (*e.g.*, number of packets/bytes matched)
- Action: to be performed after a match
- Timeouts: idle or a hard timeout before flow entry is evicted
- Cookie: set by the controller to identify a flow entry
- Flags: determine behavior wrt. a flow entry (*e.g.*, notify controller when entry is evicted)

Flow Table Structure



- Ingress Port: The identifier of the port where packet was received
- Layer 2 Headers: Source/Destination MAC addresses, Ethernet type etc.)
- Layer 3 Headers: Source/Destination IP addresses, IP protocol *etc.*
- Layer 4 Headers: TCP/UDP source/destination ports
- VLAN ID: Virtual LAN tag number
- Meta-data: Additional information passed from a previous flow table (when flow tables are pipelined)
- Many more (OpenFlow 1.4 defines 41 match fields)

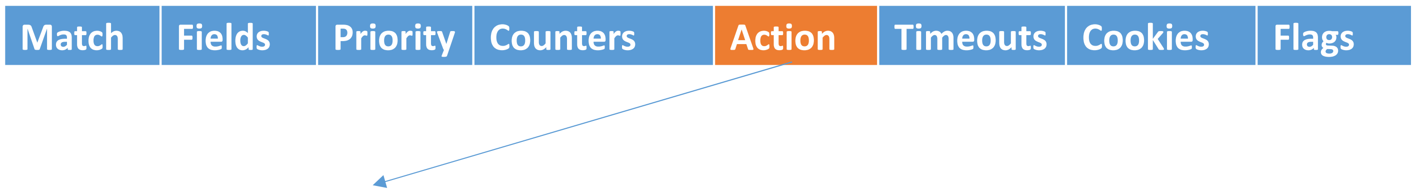
Flow Table Structure



Counters are associated with flow table entries, the flow table itself,

- queues, groups, etc.
- reference count: number of active entries in flow table
- duration: duration a flow table entry is in TCAM
- received/sent packets: per port counter
- matched packets/bytes: number of packets/bytes matching a flow table entry
- ! many more…

Flow Table Structure



- Output: send packet to a specific port
- Goto: continue processing at a different flow table
- Meter: redirect packet to a meter table for QoS enforcement
- Group: process packet through specific group of ports
- Push/Pop tag: encapsulate/decapsulate packet to/from a tunnel header (VLAN, MPLS, GRE etc.)
- Set field: modify a header field
- Change TTL: update IPv4 TTL or IPv6 hop limit or MPLS TTL
- Drop: drops a packet

Flow Table Structure

Match	Fields	Priority	Counters	Action	Timeouts	Cookies	Flags

- idle_timeout: A flow table entry is evicted from TCAM if no packets have matched the entry in the past idle_timeout time duration setting a value of 0 disables the timer
- hard_timeout: A flow table entry is evicted from TCAM after hard_timeout period elapses from the time of installing the entry Setting a 0 value disables this timer

OpenFlow Key Messages

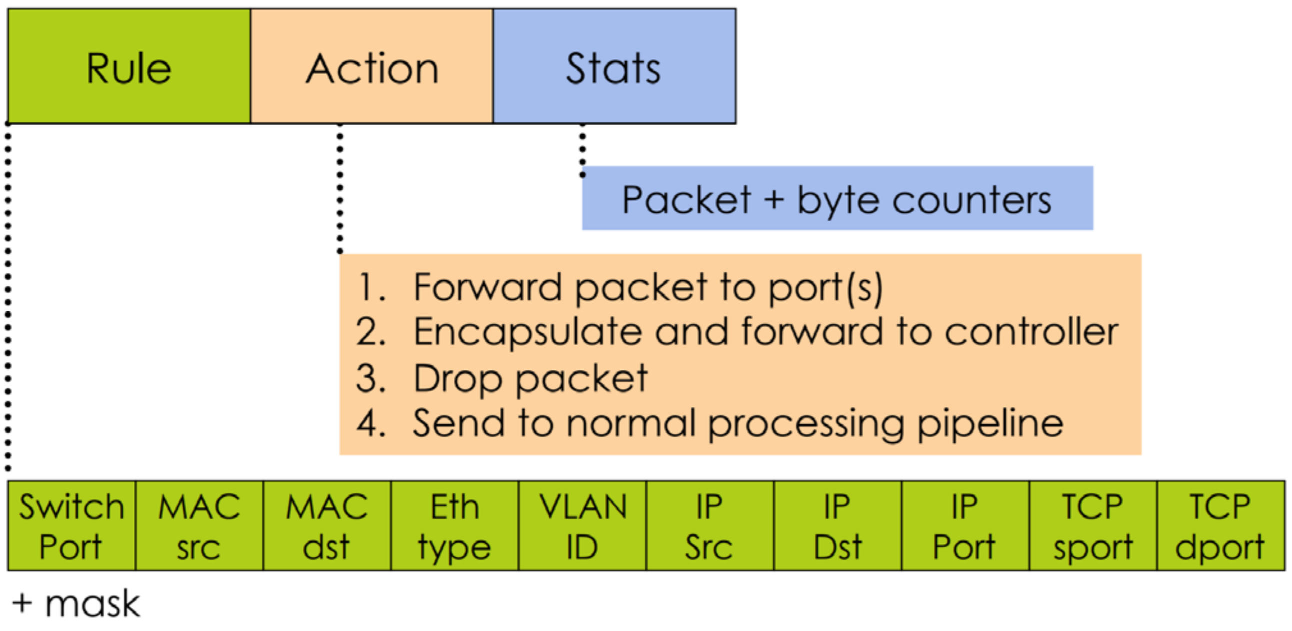
- Switch-to-controller messages
  - PacketIn: When a packet does not match any flow table entry the switch encapsulates the packet in a PacketIn message and sends it to the controller. This message is a request from the switch to the controller for path setup
  - FlowRemoved: When a flow table entry expires a FlowRemoved message is sent to the controller
  - PortStatus: A switch uses this message to inform controller about any status change of a port (e.g., port went down)

OpenFlow Key Messages (cont)

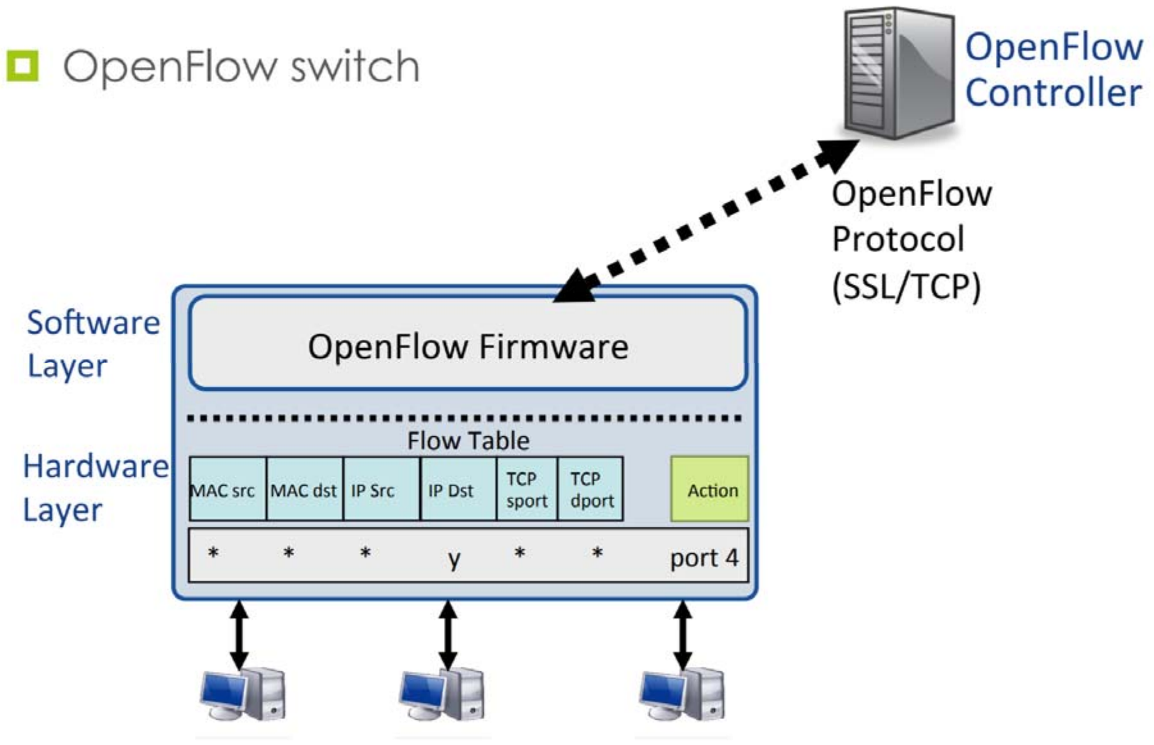
- Controller-to-switch messages
  - SetConfig: this message is used by a controller to configure switch parameters
  - FlowMod/GroupMod/PortMod: controller uses these messages to modify switch state such as flow table entry, group tables, port status etc.
  - PacketOut: with this message the controller asks a switch to send packet out of a specific switch port

- Symmetric Messages
  - GetConfigReq/GetConfigRes: these messages allow synchronous communication between switch and controller to exchange current configuration state
  - FeatureReq/FeatureRes: these messages are used by a controller to query the features supported by a switch
  - StatReq/StatRes: these messages allow controller to read statistics (packet count, byte count, errors etc.) from the switch. Statistics can be per port, per table or per flow entry

OpenFlow Flow Table Entry



OpenFlow in action

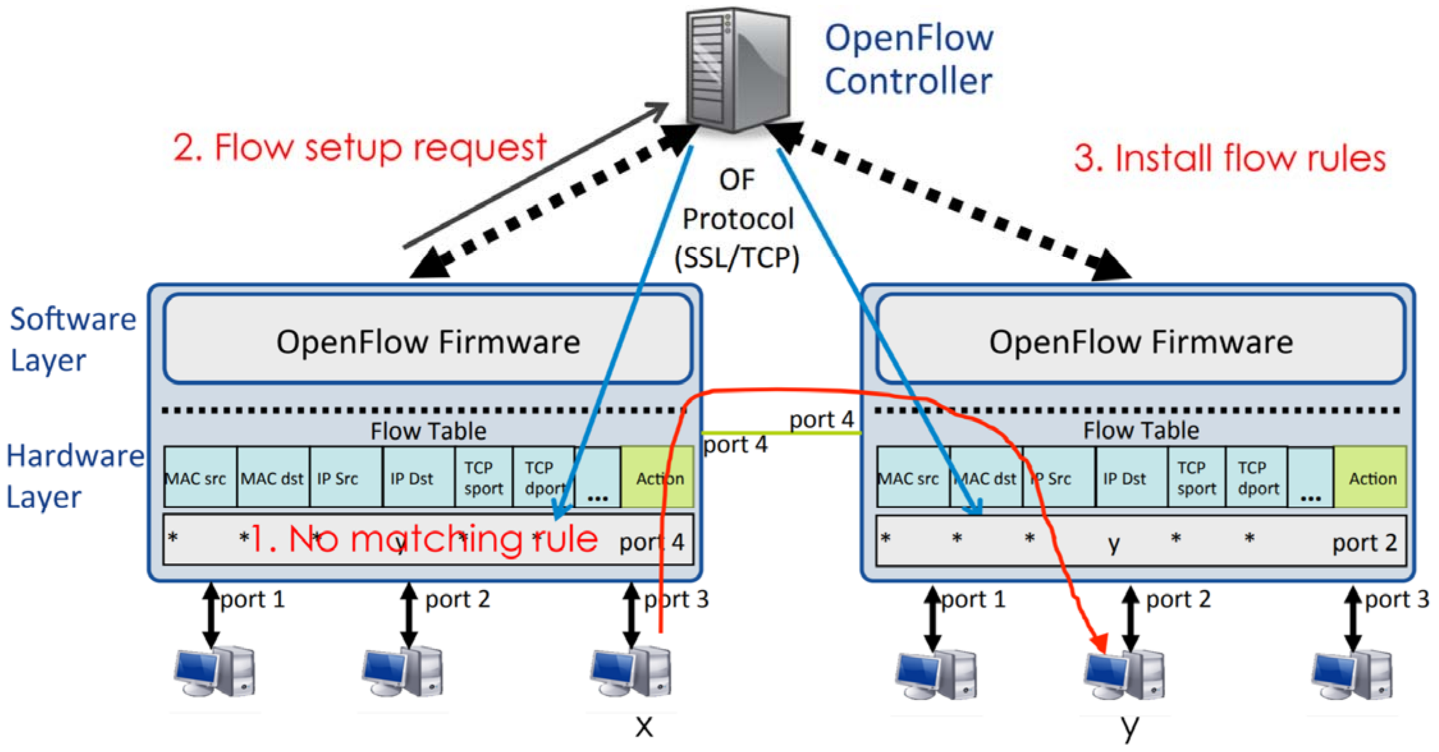


OpenFlow data plane abstraction

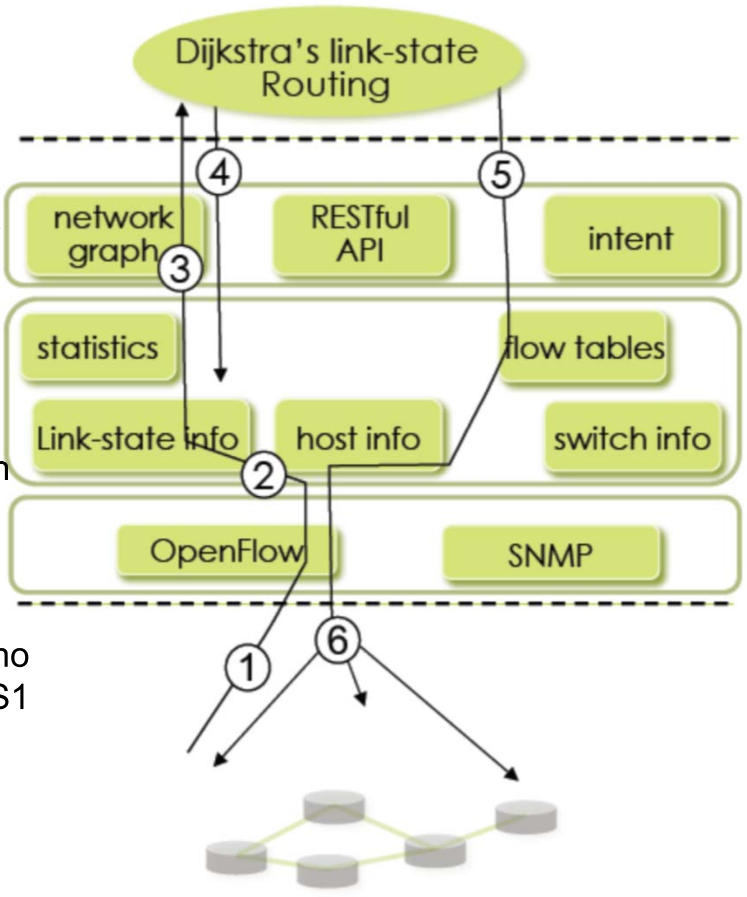
- *(match+action):* unifies a wide range of data plane functions
- Router
  - *match*: longest destination IP prefix
  - *action*: forward on an output link
- Switch
  - *match*: destination MAC address
  - *action*: forward or flood

Firewall
match: IP addresses and TCP/UDP port numbers
action: allow or deny

NAT
match: IP address and port
action: rewrite address and port


OpenFlow Controller
2. Flow setup request
3. Install flow rules
OF Protocol (SSL/TCP)
Software Layer
Hardware Layer
OpenFlow Firmware
Flow Table
MAC src
MAC dst
IP Src
IP Dst
TCP sport
TCP dport
...
Action
* * * y * * port 4
1. No matching rule
port 4
port 4
OpenFlow Firmware
Flow Table
MAC src
MAC dst
IP Src
IP Dst
TCP sport
TCP dport
...
Action
* * * y * * port 2
port 1
port 2
port 3
port 1
port 2
port 3
x
y

OpenFlow in Action: Flow Setup



1) S1, receives a packet and there is no forwarding rule matching the packet. S1 sends a PacketIn message to controller

2) SDN controller receives PacketIn message, identifies source and destination

3) Dijkstra's routing algorithm application has previously registered to be called when PacketIn arrives. It is called in this step.

4) Dijkstra's routing algorithm access network graph info, link state info in controller, computes new routes

5) link state routing app interacts with flow-table computation component in SDN controller, which computes new flow tables needed

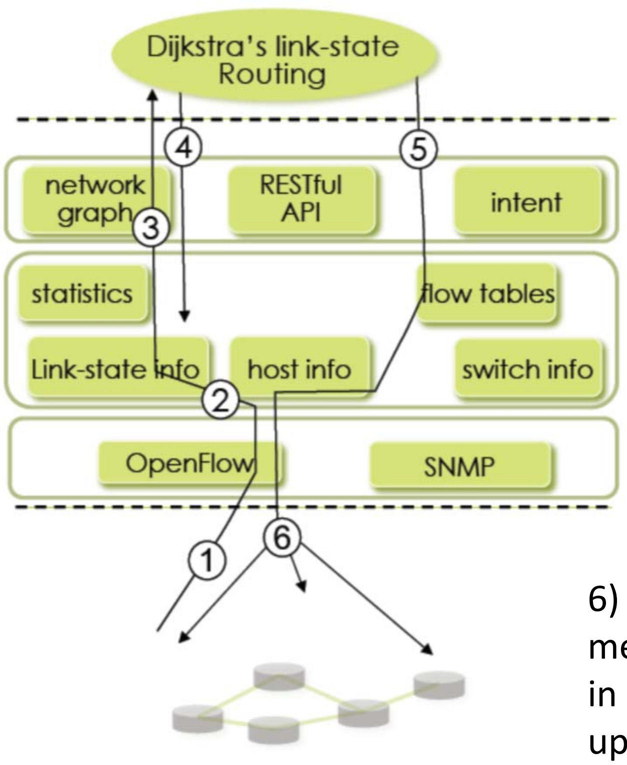
6)Controller uses FlowMod message to install new tables in switches that need updating

OpenFlow in Action: Failure Handling

3)Dijkstra's routing algorithm appl has previously registered to be called whenever a link status changes. It is called.

2) SDN controller receives OpenFlow message, updates link status info

1) S1, experiencing link failure uses OpenFlow PortStatus message to notify controller



4) Dijkstra's routing algorithm accesses network graph and link state info in controller, computes new routes

5) link state routing app interacts with flow-tablecomputation component in SDN controller, which computes new flow rules as needed

6) Controller uses FlowMod message to install new rules in switch tables that need updating

Limitations of OpenFlow Data Plane

- SDN control and data planes are still coupled to some extent
  - What can be matched (i.e., set of packet headers) and what can be done (i.e., set of actions) are still tied to hardware implementation
  - Number of OpenFlow match fields increased from 12 (OpenFlow v1.0) to 41 (OpenFlow v1.4).
    - Hardware support for all the fields is still falling behind the specifications
- OpenFlow is playing catch-up as new protocols emerge

Possible Solutions

- Reconfigurable hardware (e.g., FPGA)
  - Not production ready yet
- Programmable Switch ASIC (e.g., Intel FlexPipe)
  - Low-level proprietary APIs and restricted programmability
- Pure software-switching
  - only at servers
  - number of physical ports hard to scale in servers

Ideal SDN Data Plane

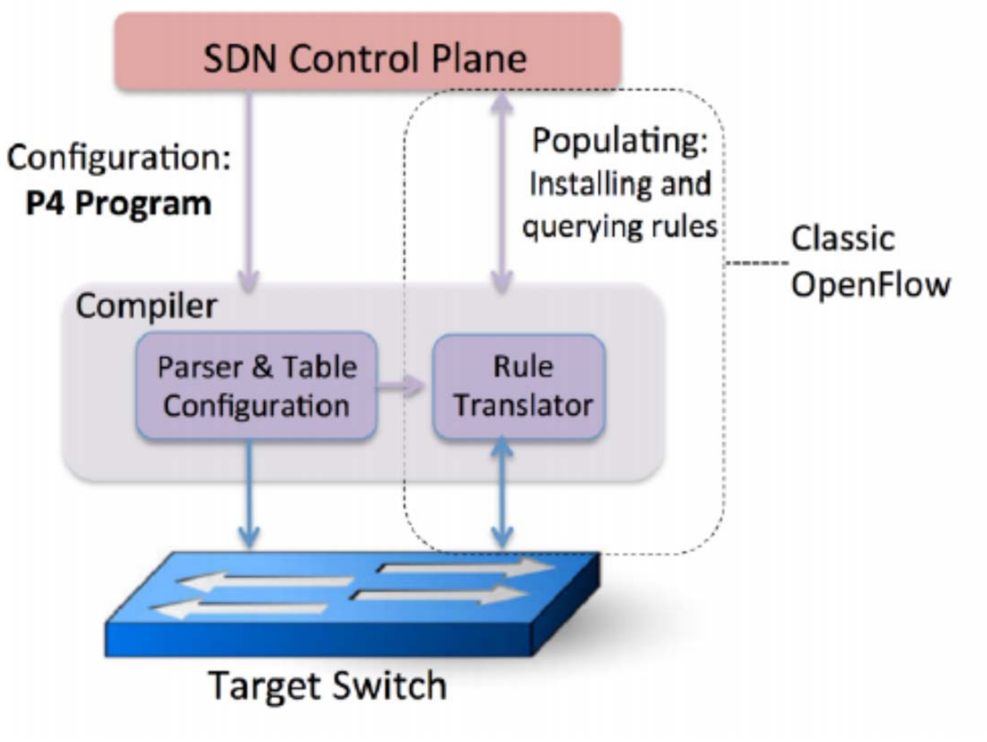
- Flexible and reconfigurable packet matching
- Customizable actions
- Support for wide range of packet processing pipelines
- No hardware dependencies

P4

- Programming Protocol Independent Packet Processors
- language for expressing how packets are processed by the data plane of a programmable network element
- P4 programs specify how the various programmable blocks of a target switch architecture are programmed and connected
- P4 is:
  - Reconfigurable: allows remote controller to reconfigure packet parsing
  - Protocol independent: P4 compiler allows definition of arbitrary data areas in the packet for further processing
  - Target independent: compiler hides underlying hardware specifics from data plane specification interface

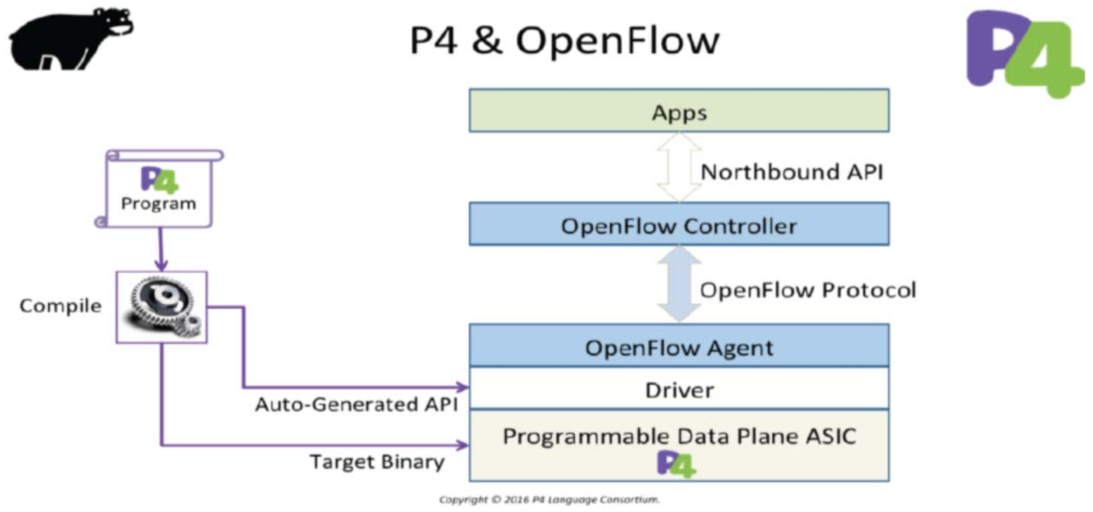
P4 Overview

- P4 compiler takes a packet processing specification and compiles it considering features of a target hardware platform
- The target hardware is then reconfigured to support packet processing according to the compiled specification
- A Control channel allows on-thefly modification of packet processing pipeline/packet parsing configuration



P4 with OpenFlow

- ❖ P4 can work with any control protocol as long as the corresponding control agent is present in the hardware
- ❖ OpenFlow is one instance of a control protocol compatible with P4
  - P4 program is written to configure target hardware to support (match+action) flow table as defined in OpenFlow
  - OpenFlow agent in target hardware communicates with controller



Hardware Support for P4

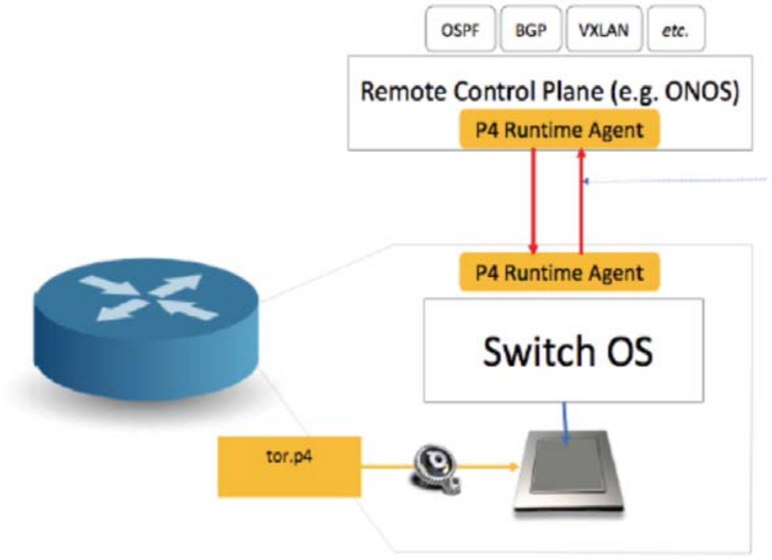
- P4 language for specifying custom packet processing is now supported by a wide range of hardware
  - Barefoot Tofino switch
  - Netronome NIC
  - Xilinx FPGA
  - PISCES software switch (based on Open vSwitch)

P4 Runtime : Control Plane Agonistic Southband API

- P4 switches can work with any control protocol as long as corresponding controller agent is running on the switch
- Problem: adding support for new control southbound API requires firmware update and/or vendor support
- Solution: P4 Runtime
  - P4 runtime replaces the protocol specific agent on the target hardware

P4 Runtime Overview

- P4 runtime streamlines the process of adding new control protocol
  - P4 enabled hardware runs a P4 agent which is agnostic to the control protocol
  - Controllers implement a wrapper around P4 runtime
  - Adding new protocol support does not require hardware change or firmware upgrade

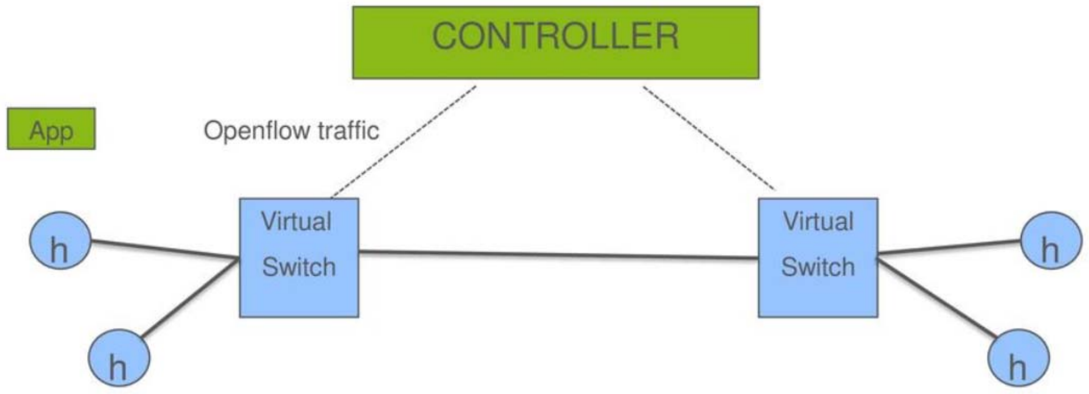


SDN Tools

- FlowVisor/OVX
  - Creates network slices that can be used & managed by different controllers (tenants)
- Mininet
  - Open source OpenFlow network emulator
- DOT
  - Distributed OpenFlow Testbed
- Oflops, cbench
  - Tools to measure performance of OpenFlow controller and switch

MININET

- Provide tools to create virtualized network with OVS
- CLI for manipulating network dynamically
- Virtualized hosts
- Controller: POX, Ryu, ODL, etc.



Segment Routing (SR)

Segment Routing enables control of flows

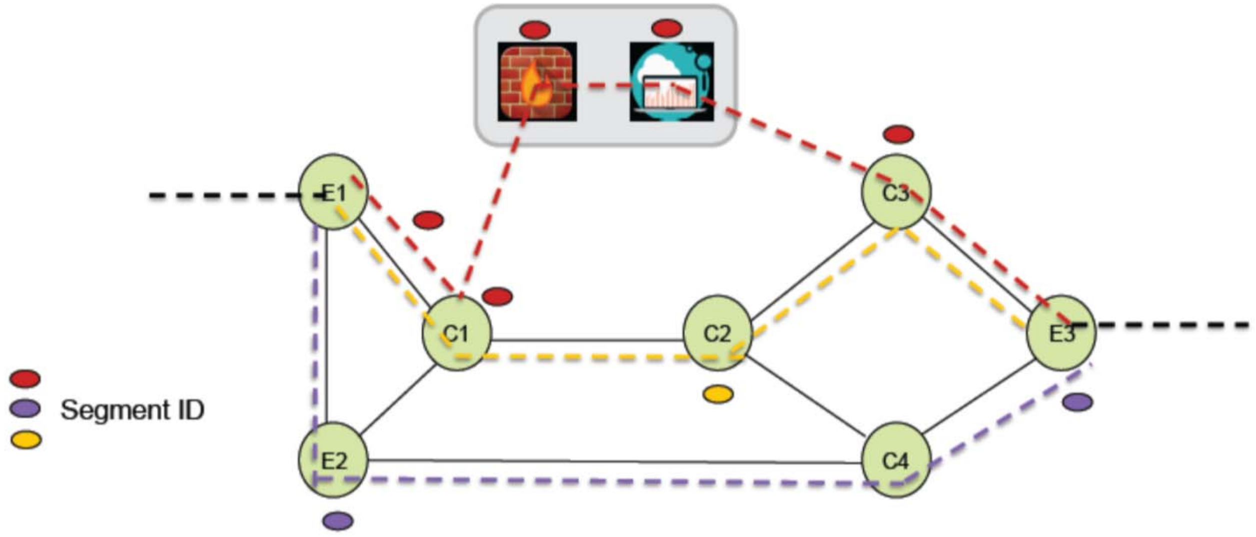
- Next generation traffic engineering mechanism
- Source-based routing
- Embed path (instructions) into packet header
- It replaces what today are RSVP TE tunnels.

SR helps deliver SDN to the network

- Centralized construction of TE paths
- Remove state from the network
- Improving scalability, operational simplicity
- SR is managed by an IP SDN controller

How Segment Routing Works

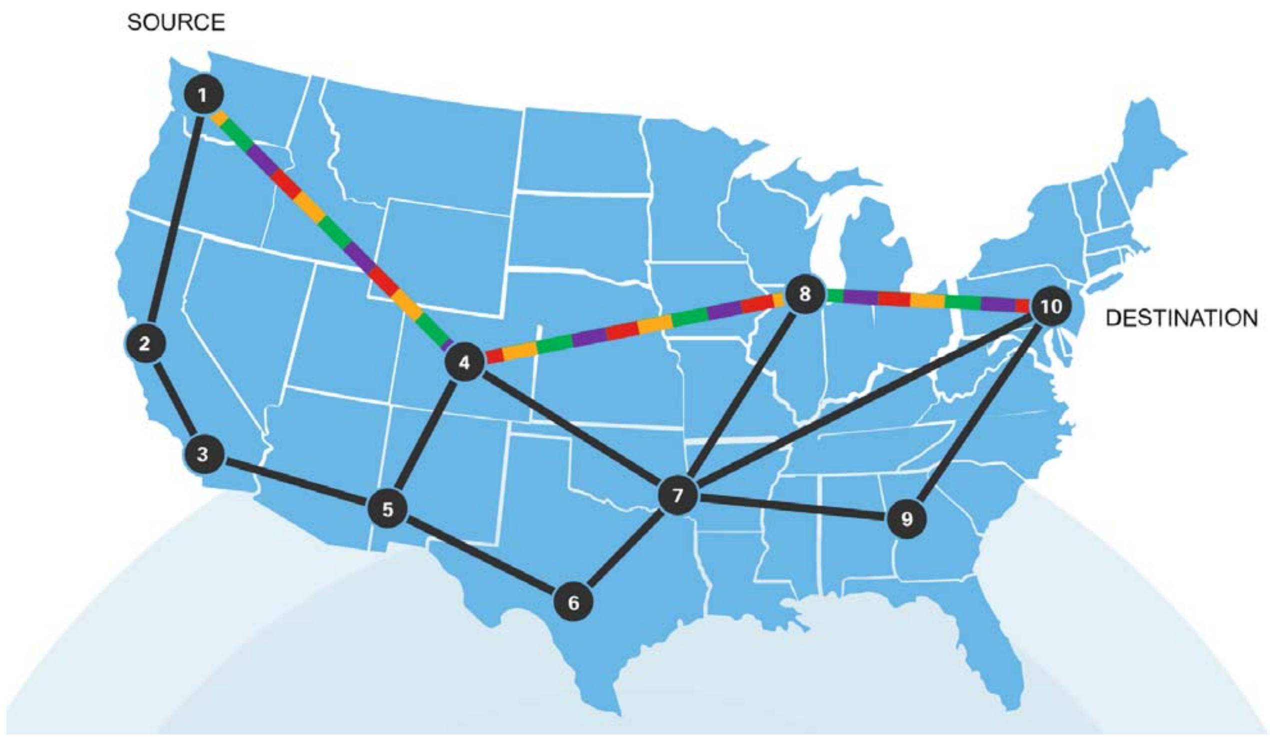
- Routing
- Traffic engineering
- Service chaining



SR Terminology

- An SR Segment
  - Connects two points within the SR domain
  - Can traverse one or more router hops
  - Is represented by a Segment Identifier (SID)
- A Segment Identifier (SID)
  - Identifies the path fragment that the packet follows
  - Can have node-local or domain-wide (a.k.a., global) significance

Example of SR



https://www.cisco.com/c/m/en_us/network-intelligence/service-provider/digital-transformation/how-does-segment-routing-enable-sdn.html

History of SDN

- ❖ Stanford University – Clean Slate Project
- ❖ *"…explore what kind of Internet we would design if we want to start with a clean slate and 20-30 years of hindsight."*

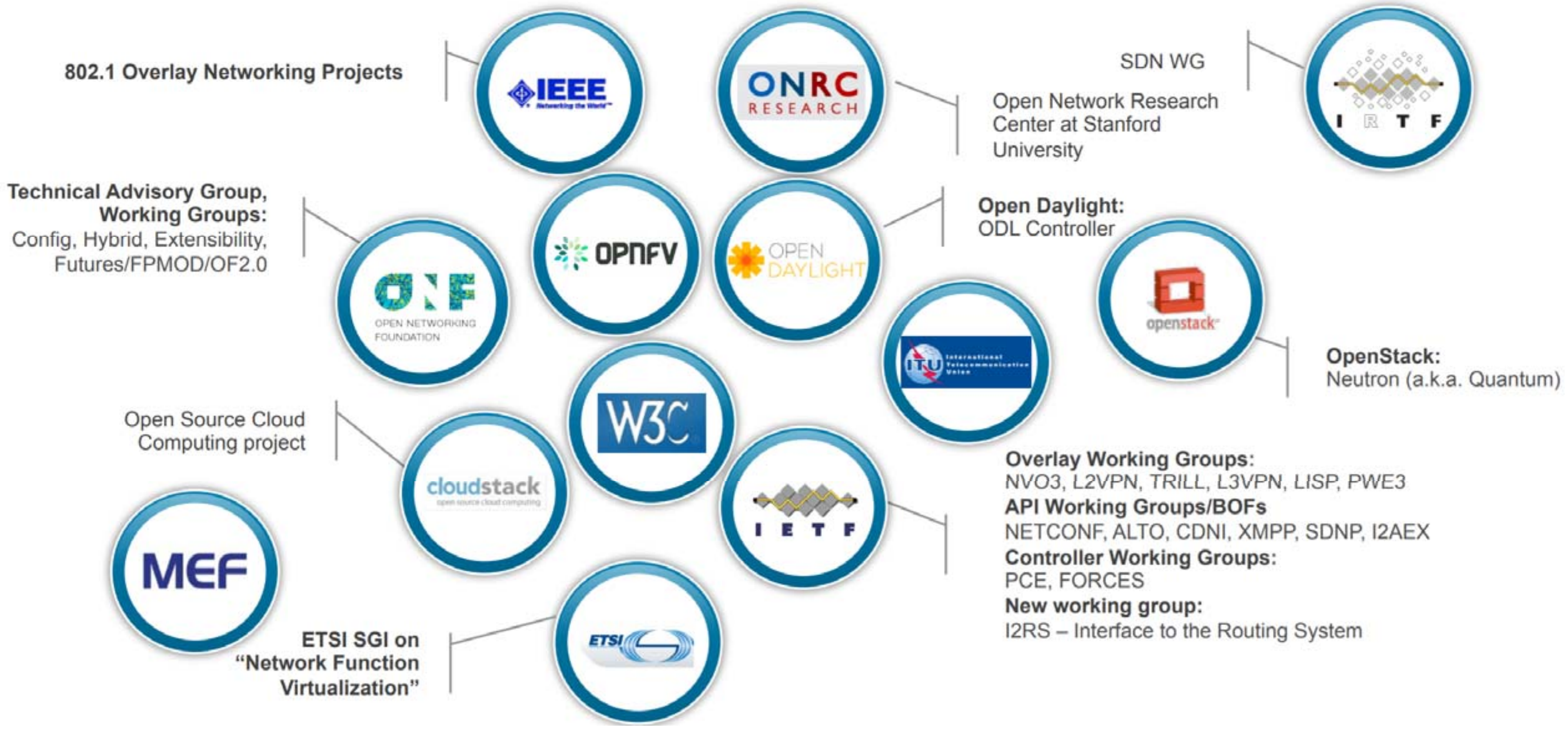
Directory > Stanford Clean Slate Program

Stanford Clean Slate Program

Stanford Clean Slate Program
Corporate Headquarters
353 Serra Street
Gates Building, Room 3A-351
Stanford, California 94305
United States
(650) 725-9077
1 - 100 employees

- ❖ Clean Slate led to the development of Openflow

Standardization Activities



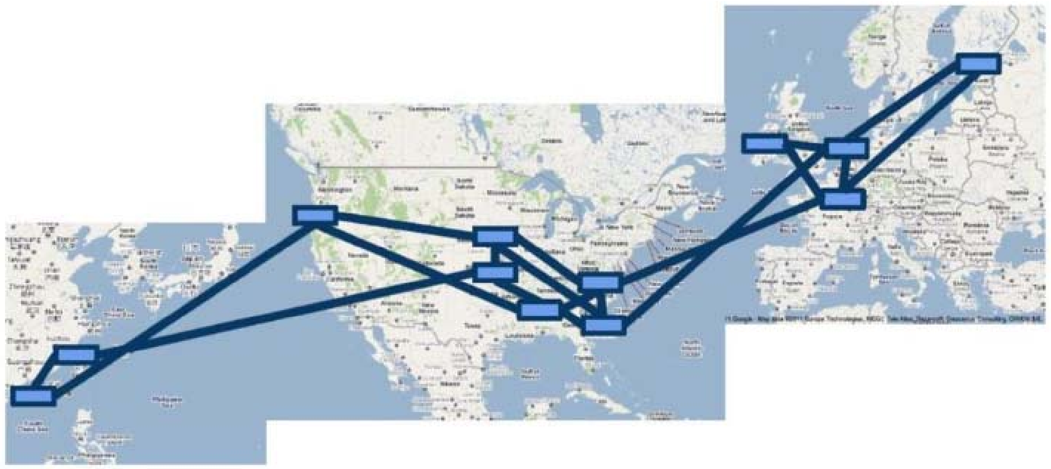
802.1 Overlay Networking Projects
IEEE
ONRC RESEARCH
SDN WG
Open Network Research Center at Stanford University
I R T F
Technical Advisory Group, Working Groups:
Config, Hybrid, Extensibility, Futures/FPMOD/OF2.0
ONF OPEN NETWORKING FOUNDATION
OPNFV
OPEN DAYLIGHT
Open Daylight:
ODL Controller
openstack
OpenStack:
Neutron (a.k.a. Quantum)
ITU International Telecommunication Union
W3C
Open Source Cloud Computing project
cloudstack
I E T F
Overlay Working Groups:
NVO3, L2VPN, TRILL, L3VPN, LISP, PWE3
API Working Groups/BOFs
NETCONF, ALTO, CDNI, XMPP, SDNP, I2AEX
Controller Working Groups:
PCE, FORCES
New working group:
I2RS – Interface to the Routing System
MEF
ETSI SGI on "Network Function Virtualization"
ETSI

SDN in WANs as a most Promising Application

- SDN mostly used in data center networks
- Possibility of SDN in a WAN gained a lot of momentum after Google deployed the B4 network
- In a WAN environment SDN is used for
  - Traffic steering
  - Achieving high link utilization
  - Implementing advanced policies
    - e.g., application specific peering between ISPs

B4

- Google's B4 network for inter data center communication
- A system to boost the utilization of inter-datacenter network
- Re-configures the network's data plane to match current traffic demand
- Avoids transient congestion during rule updates by leveraging a small amount of scratch capacity on links
  - Updates are applied in a congestion-free manner



Open Problems of SDN

- SDN Controller placement problems:
  - Defines how many controllers should be deployed in the network and where
    - How many controllers are needed?
    - How does placement affect latency?
    - Where in the topology should controller go?
- Hypervisor Placement Problem[1]:
  - A hypervisor acts as an intermediate layer between the tenant SDN controllers and their respective virtual SDN networks. How many hypervisor instances are needed? Where should the hypervisor instances be placed in the network?
- Rule Placement Problem[2]:
  - A rule placement solution defines which rules must be deployed in the network and where

Blenk, A. et al. "Survey on Network Virtualization Hypervisors for Software Defined Networking", Communications Surveys & Tutorials, IEEE, Firstquarter 2016 [2] Nguyen, Xuan-Nam, et al. "Rules Placement Problem in OpenFlow Networks: a Survey." (2016)

Research Direction

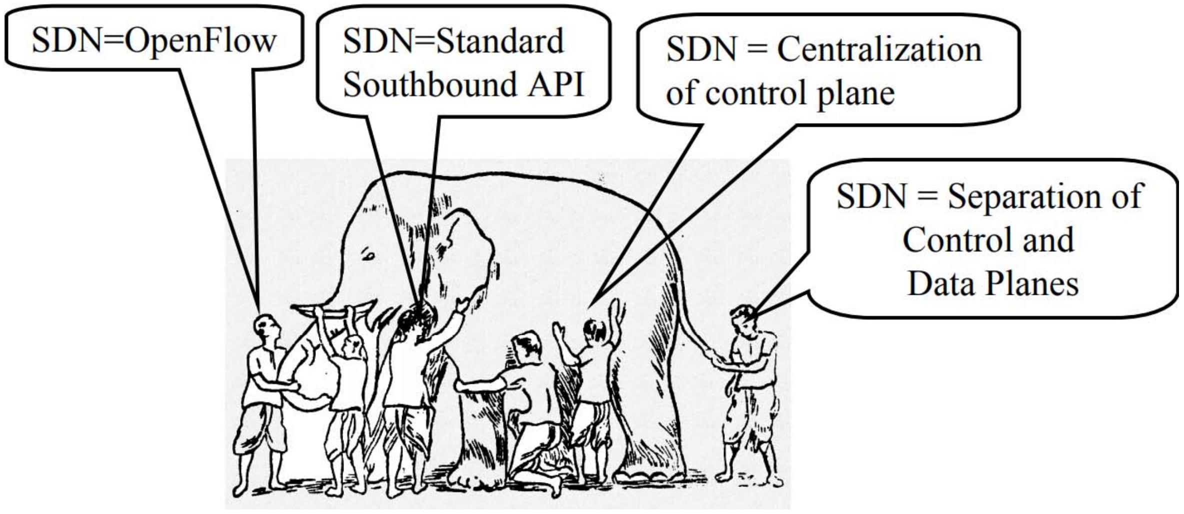
- Understanding the tradeoff between different architectures
  - Centralized vs. distributed vs. functionality offloading
- Reactive vs. proactive policies
  - Pre-install rules to reduce flow setup requests
- Ensuring high availability
  - For controller, switch and link failures
- Managing device status and statistics
  - Consistency model
  - Storage model
- Delay in SDN
  - Controller to switch propagation latency

Research Directions

- Tradeoff between monitoring accuracy & monitoring traffic volume
- Active vs. passive monitoring
  - How to extend OpenFlow monitoring features
  - Are the packet and byte counters enough?
  - Lack of proper monitoring APIs for
  - Application aware network control plane
  - Network aware applications ! or both
- Traffic Engineering (TE) is done for different objectives
  - Maximizing network utilization
  - Ensuring QoS
  - Load balancing
  - Minimizing power consumption

SDN is ….?

- All of these are mechanisms.
  - SDN is not a mechanism.
  - It is a framework to solve a set of problems
  - Many solutions


SDN=OpenFlow
SDN=Standard Southbound API
SDN = Centralization of control plane
SDN = Separation of Control and Data Planes

References

- B. A. A. Nunes, M. Mendonca, X. Nguyen, K. Obraczka and T. Turletti, "A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks," in *IEEE Communications Surveys & Tutorials*, vol. 16, no. 3, pp. 1617-1634, Third Quarter 2014.
- T. Huang, F. R. Yu, C. Zhang, J. Liu, J. Zhang and Y. Liu, "A Survey on Large-Scale Software Defined Networking (SDN) Testbeds: Approaches and Challenges," in *IEEE Communications Surveys & Tutorials*, vol. 19, no. 2, pp. 891-917, Secondquarter 2017.
- Yili Gong , Wei Huang , Wenjie Wang, Yingchun Lei, A survey on software defined networking and its applications, Frontiers of Computer Science December 2015, Volume 9, Issue 6, pp 827–845
- X. Huang, S. Cheng, K. Cao, P. Cong, T. Wei and S. Hu, "A Survey of Deployment Solutions and Optimization Strategies for Hybrid SDN Networks," in *IEEE Communications Surveys & Tutorials*.
- T. Dargahi, A. Caponi, M. Ambrosin, G. Bianchi and M. Conti, "A Survey on the Security of Stateful SDN Data Planes," in *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1701-1725, thirdquarter 2017.

- T. Dargahi, A. Caponi, M. Ambrosin, G. Bianchi and M. Conti, "A Survey on the Security of Stateful SDN Data Planes," in *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1701-1725, thirdquarter 2017.

- R. Amin, M. Reisslein and N. Shah, "Hybrid SDN Networks: A Survey of Existing Approaches," in *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 3259-3306, Fourthquarter 2018.
- Q. Yan, F. R. Yu, Q. Gong and J. Li, "Software-Defined Networking (SDN) and Distributed Denial of Service (DDoS) Attacks in Cloud Computing Environments: A Survey, Some Research Issues, and Challenges," in *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 602-622, Firstquarter 2016.
- S. Scott-Hayward, S. Natarajan and S. Sezer, "A Survey of Security in Software Defined Networks," in *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 623-654, Firstquarter 2016.
- I. Ahmad, S. Namal, M. Ylianttila and A. Gurtov, "Security in Software Defined Networks: A Survey," in *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2317-2346, Fourthquarter 2015.

References

- S.H. Yeganeh, et al. "Kandoo: a framework for efficient and scalable offloading of control applications", Proc. of ACM HotSDN 2012, pp. 19 – 24
- T. Koponen, et al. "Onix: A distributed control platform for large-scale production networks", In Proc. Of USENIX OSDI 2010
- M. Yu, et al. "Scalable flow-based networking with DIFANE." ACM SIGCOMM Computer Communication Review 40(4): 351-362, 2010
- N. Handigol, et al. "I Know What Your Packet Did Last Hop: Using Packet Histories to Troubleshoot Networks", In Proc. of USENIX NSDI 2014, pp. 71-85
- A. Khurshid, et al. "Veriflow: Verifying network-wide invariants in real time", In Proc. of USENIX NSDI 2013
- C. Yu, et al. "Flowsense: Monitoring network utilization with zero measurement cost", Proc. of PAM 2013: 31-41
- S.R. Chowdhury, et al. "Payless: A low cost network monitoring framework for software defined networks", In Proc. of IEEE/IFIP NOMS 2014, pp. 1-9

References

- N. Handigol, et al. "Aster*x: Load-Balancing as a Network Primitive", In Proc. of ACLD 2010 (Poster)
- S. Jain, et al. "B4: Experience with a globally-deployed software defined WAN", ACM SIGCOMM Computer Communication Review 43(4): 3-14, 2013
- J. Kurose, and K.W. Ross. "Computer networking: a top-down approach." (7th Edition)
- https://opencord.org/
- https://wiki.onosproject.org/
- "P4 Runtime",
- https://p4.org/api/p4-runtime-putting-the-control-plane-in-charge-of-the-forwarding-plane.html
- "P4 and OpenFlow", https://p4.org/p4/clarifying-the-differences-between-p4-and-openflow.html
- P. Bosshart et al., "P4: Programming Protocol-Independent Packet Processors", ACM SIGCOMM Computer Communication Review