

Review of Previous Lecture

- What is the control plane of a network?
 - The functions in the network that control the behavior of the network, e.g., network paths, forwarding behavior
- What is the data plane of a network?
 - The functions in the network that are responsible for forwarding or not forwarding traffic. Typically, the data plane is instantiated as forwarding tables in routers, switches, firewalls, and middleboxes

Review of Previous Lecture (Cont'd)

- Which network first had the separation of control plane and data plane?
 - The telephone network, specifically AT&T introduced then Network Control Point in 1981 to support a wide range of network applications such as 800 Service and Calling Card Service.
- Why separate control?
 - More rapid innovation: control logic is not tied to hardware
 - Network wide view: easier to infer and reason about network behavior
 - More flexibility: can introduce services more rapidly

Review of Previous Lecture

- What is the definition of SDN?
 - The separation of control plane from data plane
 - A specific SDN: configuration, distribution and forwarding abstraction
- What is one API between control plane and data plane?
 - OpenFlow protocol

Outline

- Review of previous lecture
- SDN Basics
 - Concepts
 - OpenFlow
 - Controller: Floodlight
 - OF-Config
 - Mininet

SDN in a Nutshell

- SDN is defined *precisely* by these three abstractions
 - Distribution, forwarding, configuration
- SDN not just a random good idea...
 - Fundamental validity and general applicability
- SDN may help us *finally* create a discipline
 - Abstractions enable reasoning about system behavior
 - Provides environment where formalism can take hold....
- OK, but what are these abstractions?

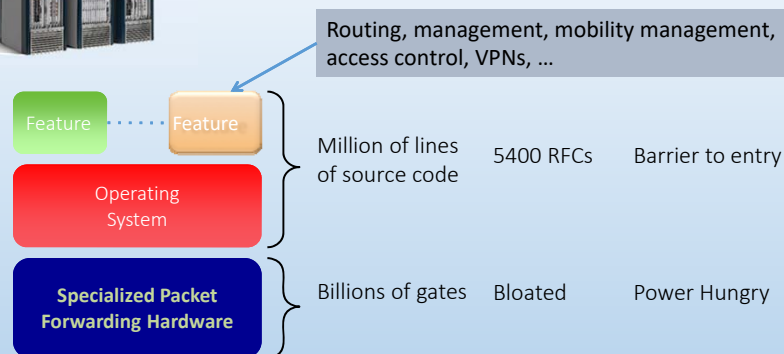
OpenFlow

- Why OpenFlow?
- How does OpenFlow work?

Why OpenFlow?



The Ossified Network(2007)



Many complex functions baked into the infrastructure

*OSPF, BGP, multicast, differentiated services,
Traffic Engineering, NAT, firewalls, MPLS, redundant layers, ...*

An industry with a “mainframe-mentality”, reluctant to change

Little ability for non-telco network operators to get what they want

Functionality defined by standards, put in hardware, deployed on nodes

Research Unproductivity

Faster networks but not better networks

- Lots of *deployed* innovation in other areas
 - Operating Systems: various filesystems, schedulers, virtualization
 - Distributed Systems: DHTs, CDNs, MapReduce
 - Compilers: JITs, new language paradigms
 - Services and applications: Cloud computing, Big Data, Social Networks, ...
- Networks are largely the same as years ago
 - Ethernet, IPv4, WiFi –IPv6 took ages
- Rate of change of the network seems slower in comparison
 - Need better tools and abstractions to demonstrate and deploy
- Researchers need flexible and changeable platforms for experimentation

Another Problem: Closed Systems (Vendor Hardware)

- Can't extend –vertically integrated
- Stuck with interfaces (CLI, SNMP, etc)
- Hard to meaningfully extend
- Hard to meaningfully collaborate
- Need a fully open system – a Linux equivalent

Future Internet Research

- What should be the architecture of a future Internet?
 - – Beyond IPv4 and IPv6
- Future Internet Architecture (FIA) project by the US National Science Foundation (2010)
- Global Environment for Network Innovations (GENI) 2006
- Future Internet and Experimentation (FIRE) Program by the European Commission's 7th Framework Program (2006)
- Clean Slate Approach

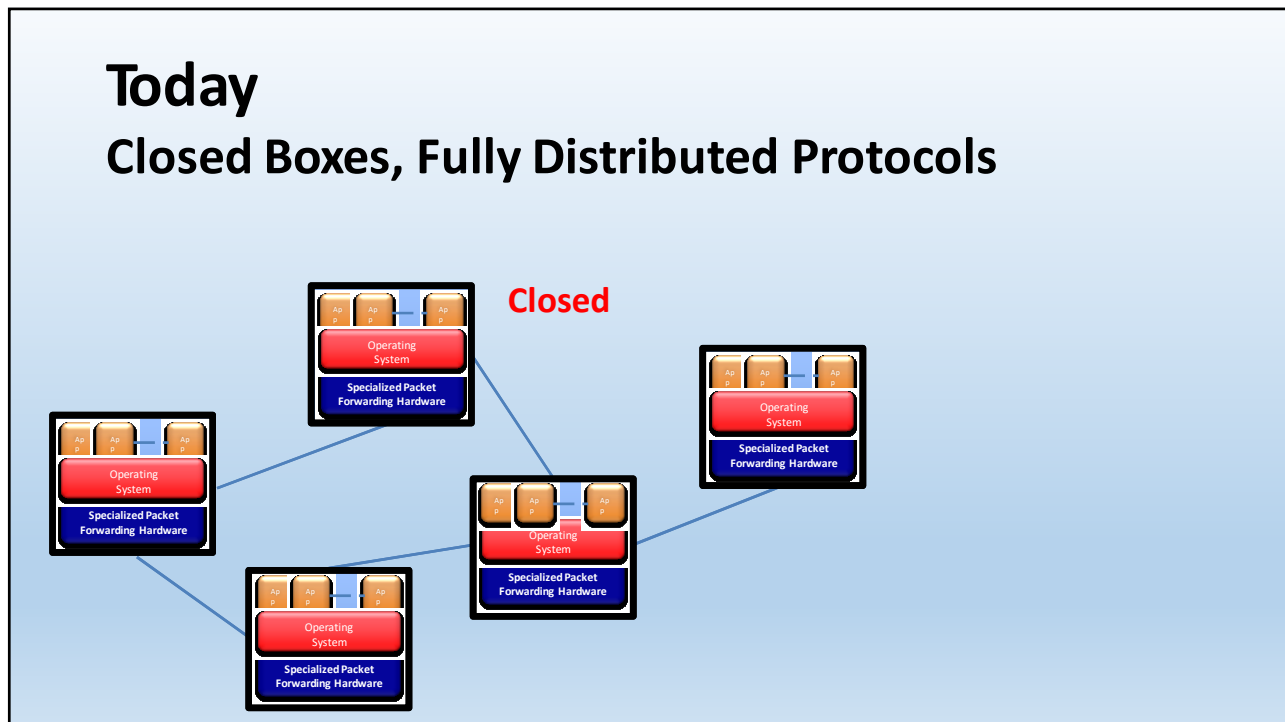
The networking researcher's playground

Approach	Example	Performance Fidelity	Scale	Real User Traffic?	Complexity	Open
"ideal system"	n/a	High	High	Yes	Low	Yes
Simulation	NS-2, NS-3, Opnet	medium	medium	no	medium	yes
Emulation	Mininet	medium	low	no	medium	yes
Software Switches	Linux box, Clickrouter	poor	low	yes	medium	yes
HW impl.	NetFPGA	high	low	yes	high	yes
Network Processors	Intel IXP family	high	medium	yes	high	yes
Vendor Switches		high	high	yes	low	No

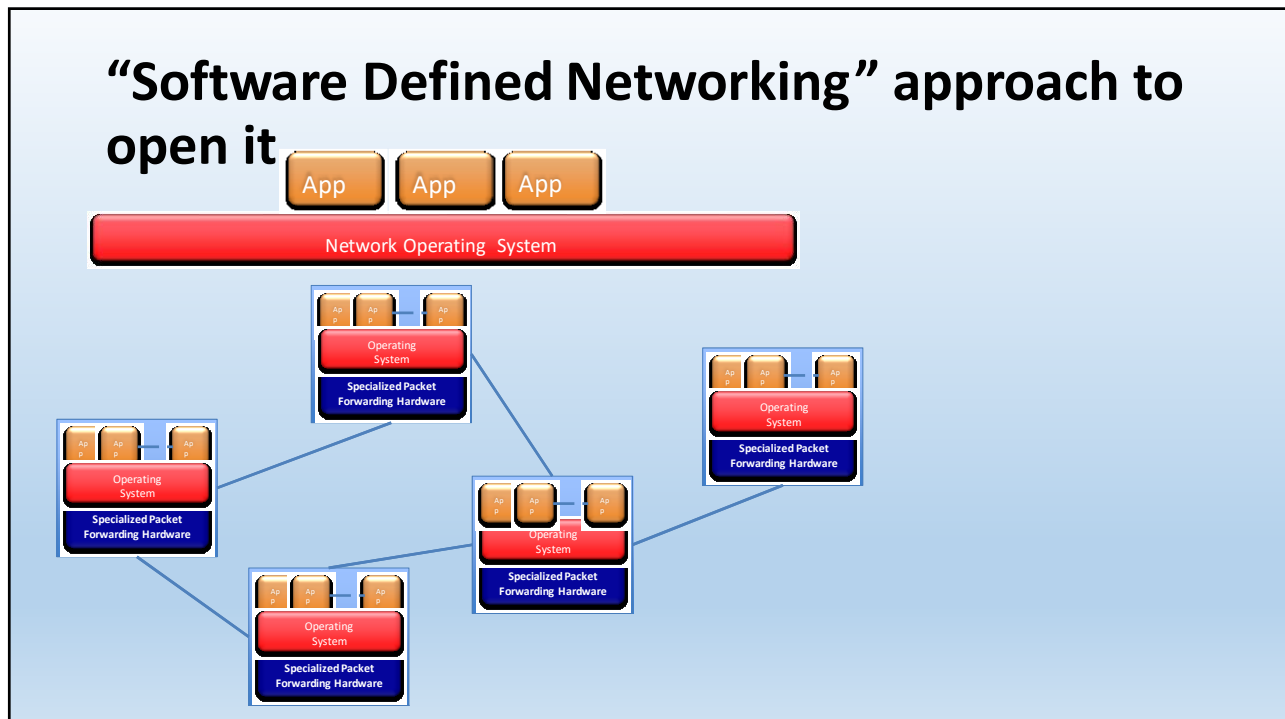
So let's open up the vendor switches!

Today

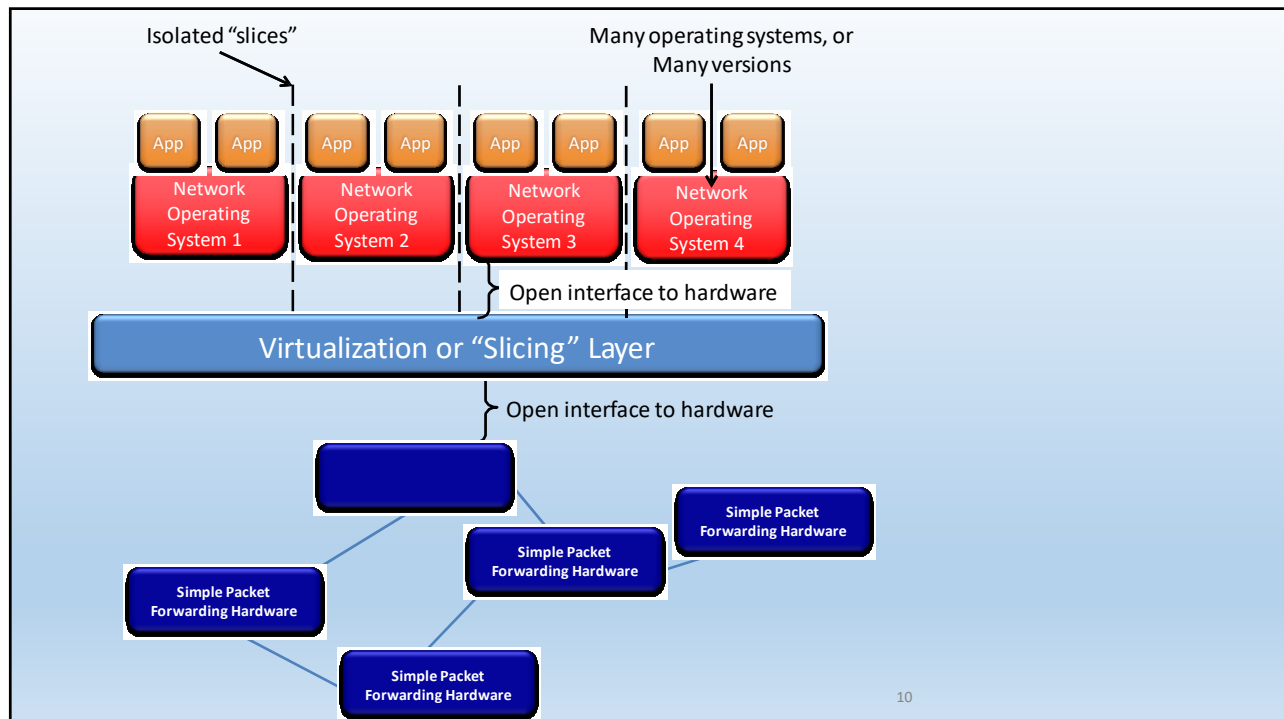
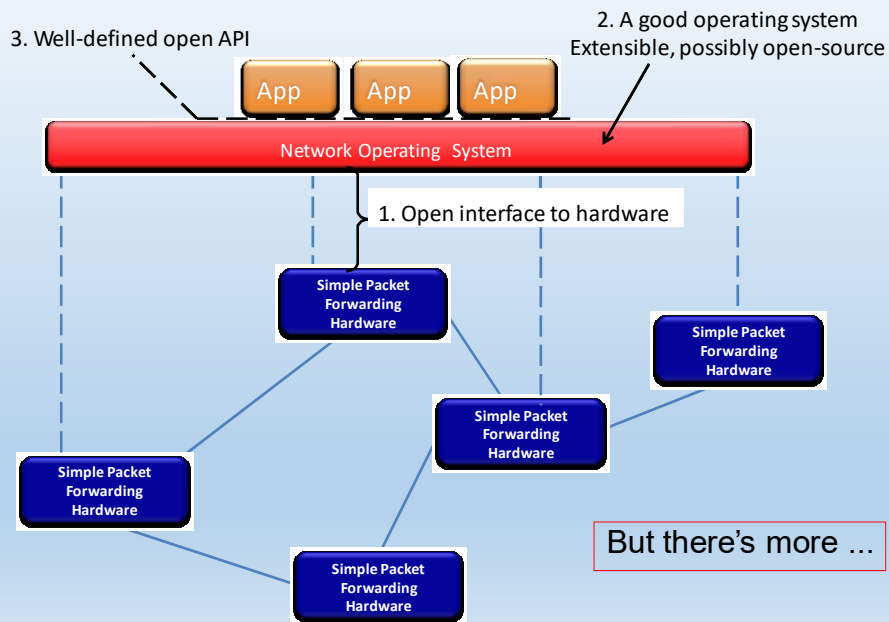
Closed Boxes, Fully Distributed Protocols



“Software Defined Networking” approach to open it



The “Software-defined Network”



Research questions

- How to design the interfaces
 - To the hardware («southbound»)
 - API of the network operating system («northbound»)
- Design of the virtualization layer
- Design of the network operating system
- How to achieve perfect isolation between different slices
- How to develop applications (network programming language?)
- What about security? Attack surface increased/decreased? Secure app development?

Abstractions

- Network control planes need abstractions
 - Abstractions solve architectural problems and enable evolvability
 - Today's layers (L2, L3, ..) are good data path abstractions but not useful for control interfaces
- Networks work because we can master complexity
 - but what we need are the right abstractions allowing for simple designs

Programming Made the Transition

- Machine Languages: no abstractions
 - Higher-level languages, OS + other abstraction
 - files, virtual memory, functions, data structures, ...
 - Modern languages: even more abstractions
 - objects, garbage collection, threads, locks, ...
 - Abstractions simplify programming: they make it easier to write, maintain, and reason about programs.
- Could networking follow this same path?

Forwarding Abstraction

- **Forwarding behavior** specified by a control program.
- Possibilities:
 - Any C++ or x86 program implementing forwarding
 - strategies
 - OpenFlow is one example of a forwarding abstraction.

State Distribution Abstraction

- Control programs should not have to handle distributed-state details
- Proposed abstraction: global network view
- Control program to operate on network view
 - Input: global network view (graph)
 - Output: configuration of each network device
- Network OS provides network view
 - Routing becomes a simple task: Just use Dijkstra's algorithm to find the shortest path (no need for distributed routing alg. like RIP/OSPF)

Specification Abstraction

- Give control program abstract view of network
- Provide enough detail to specify goals, but not to implement them
- Define the policies (specification)
- Use a “compiler” to generate the implementation

SDN has a few relatives

- Key ideas present in some form in
 - [Active Networks](#) (1996 -): Capsules and *programmable networks* approaches
 - [IETF ForCES](#) Working Group (2003): *Forwarding and Control Element Separation*
 - [4D](#) (2005): A new Approach to Network Control and Management, *proposing a network-wide view*
 - [Sane](#) (2006): Protection layer, *logically centralized*
 - Signaling System No. 7 (SS7): A *control plane* for the PSTN
 - [Ethane](#) (2007): The ancestor of *OpenFlow*

SDN has a few relatives

- Key ideas present in some form in
 - [Active Networks](#) (1996 -): Capsules and *programmable networks* approaches
 - [IETF ForCES](#) Working Group (2003): *Forwarding and Control Element Separation*
 - [4D](#) (2005): A new Approach to Network Control and Management, *proposing a network-wide view*
 - [Sane](#) (2006): Protection layer, *logically centralized*
 - Signaling System No. 7 (SS7): A *control plane* for the PSTN
 - [Ethane](#) (2007): The ancestor of *OpenFlow*

SDN has a few relatives

- Key ideas present in some form in
 - [Active Networks](#) (1996 -): Capsules and
 - programmable networks approaches
 - [IETF ForCES Working Group \(2003\): Forwarding](#)
 - and Control Element Separation
 - [4D](#) (2005): A new Approach to Network Control and Management, proposing a network-wide view
 - [Sane](#) (2006): Protection layer, logically centralized
 - Signaling System No. 7 (SS7)
 - [Ethane](#) (2007): The ancestor of OpenFlow

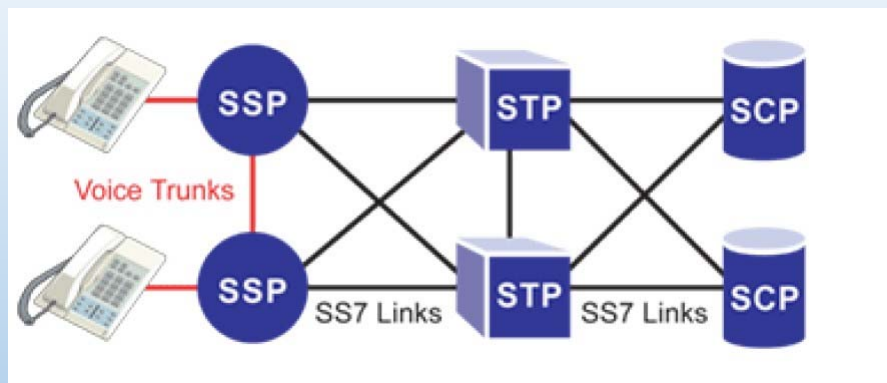
SDN has a few relatives

- Key ideas present in some form in
 - [Active Networks](#) (1996 -): Capsules and
 - programmable networks approaches
 - [IETF ForCES Working Group \(2003\): Forwarding](#)
 - and Control Element Separation
 - [4D](#) (2005): A new Approach to Network Control
 - and Management, proposing a network-wide view
 - [Sane](#) (2006): Protection layer, logically centralized
 - [Signaling System No. 7 \(SS7\)](#)
 - [Ethane](#) (2007): The ancestor of OpenFlow

Signaling System No. 7

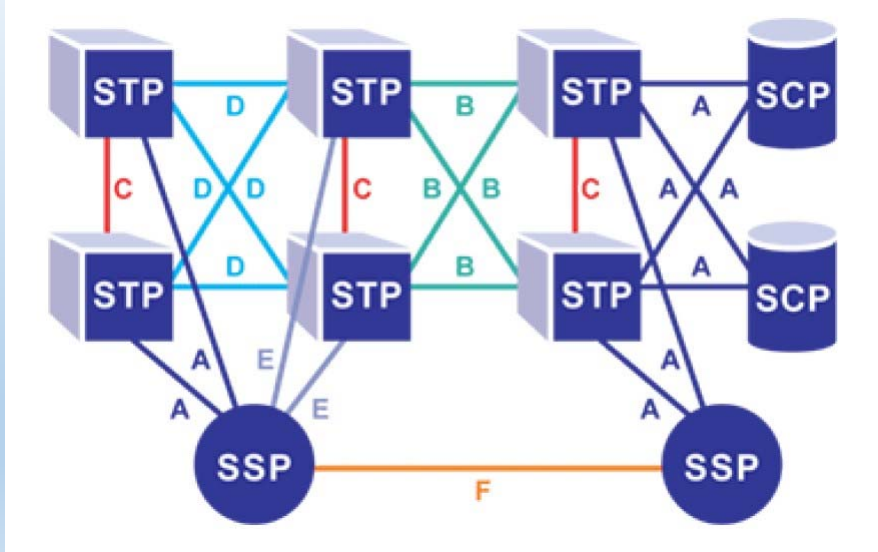
- basic call setup, management, and tear down
- wireless services such as personal communications services (PCS), wireless roaming, and mobile subscriber authentication
- toll-free (800/888) and toll (900) wireline services
- enhanced call features such as call forwarding, calling party name/number display...
- efficient and secure worldwide telecommunications

SS7 basic architecture



SSP (Service Switching Point)
STP (Signal Transfer Point)
SCP (Service Control Point)

SS7 link types and redundant layout



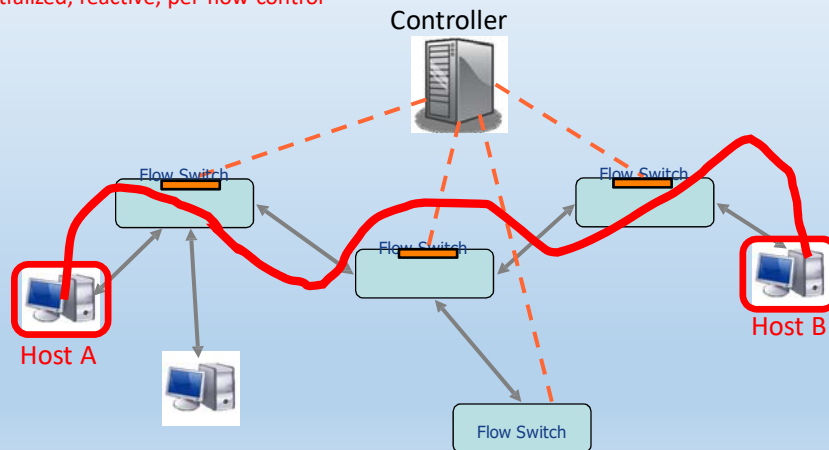
All labeled links carry control messages only

SDN has a few relatives

- Key ideas present in some form in
 - [Active Networks](#) (1996 -): Capsules and
 - programmable networks approaches
 - [IETF ForCES](#) Working Group (2003): Forwarding
 - and Control Element Separation
 - [4D](#) (2005): A new Approach to Network Control and Management, proposing a network-wide view
 - [Sane](#) (2006): Protection layer, logically centralized
 - Signaling System No. 7 (SS7)
 - [Ethane](#) (2007): The ancestor of OpenFlow

Ethane, a precursor to OpenFlow

Centralized, reactive, per-flow control

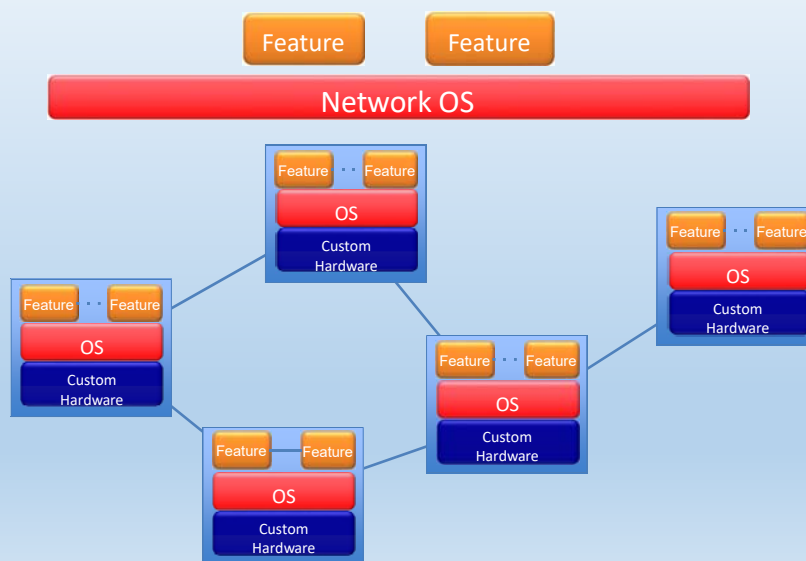


See [Ethane SIGCOMM 2007](#) paper for details

OpenFlow: a pragmatic compromise

- + Speed, scale, fidelity of vendor hardware
- + Flexibility and control of software and simulation
- + Vendors don't need to expose implementation
- + Leverages hardware inside most switches today (ACL tables)
- **Least-common-denominator** (stick with only features they all have in common) **interface may prevent using all hardware features**
- **Limited table sizes**
- **Switches not designed for this**
- **New failure modes to understand**
- **Security?**

Paradigm Shift : Network restructuring



Paradigm Shift : Network restructuring

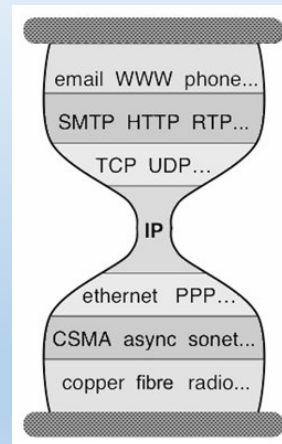
- Networks have become way too complex
- Networks are managed today by masters of complexity
- We need to extract simplicity (decompose the problem)
 - To help admins and allow easier and faster innovation
- Abstractions key to extracting simplicity
 - Abstract things : easier to understand/admin/debug
- “Modularity based on abstraction is the way things get done” (Barbara Liskov, MIT)
- We need to define Abstractions → Interfaces → Modularity

What About Networking Abstractions?

- Consider the **data** and **control planes** **separately**
 - Different tasks, so naturally different abstractions

Abstractions for Data Plane: Layers

Applications
 ...built on...
 Reliable (or unreliable) transport
 ...built on...
 Best-effort global packet delivery
 ...built on...
 Best-effort local packet delivery
 ...built on...
 Physical transfer of bits



The Importance of Layering

- **Decomposed** delivery into basic components
- **Independent**, compatible **innovation** at each layer
 - Clean “separation of concerns”
 - Leaving each layer to solve a tractable problem
- Responsible for the success of the Internet!
 - Remember : **simple & dumb**, intelligence at the edges based on layers
- Rich ecosystem of independent innovation

(Too) Many Control Plane Mechanisms

- Network management has many goals : achieving those is **the job of the control plane**
- Control Plane : variety of goals, no modularity
 - **Routing**: distributed routing algorithms
 - **Manual/scripted configuration**: ACLs, VLANs, Firewalls,...
 - **Traffic engineering**: adjusting weights, MPLS,...
- **Control Plane: mechanism without abstraction**
 - Too many mechanisms, not enough functionality

Separate Concerns with Abstractions for Control Plane

1. Be compatible with low-level hardware/software
 - Need an abstraction for general **forwarding model**
2. Make decisions based on entire network
 - Need an abstraction for **network state**
3. Compute configuration of each physical device
 - Need an abstraction that **simplifies configuration**

Abstraction #1: Forwarding Abstraction

- **Express intent independent of implementation**
 - Don't want to deal with proprietary HW and SW
- Design details concern exact nature of:
 - How to select parts of the traffic: Header matching
 - And what to do with these parts: Actions

Abstraction #2: Network State Abstraction

- Abstracts away various distributed mechanisms
- Abstraction: **global network view**
 - Natural instantiation: Annotated network graph
- Enables centralized algorithms to configure network

Abstraction #3: Specification Abstraction

- Control mechanism **expresses desired behavior**
 - Whether it be isolation, access control, or QoS
- It should **not** be responsible for **implementing** that behavior on physical network infrastructure
 - Requires configuring the forwarding tables in each switch

Interfaces vs. abstractions

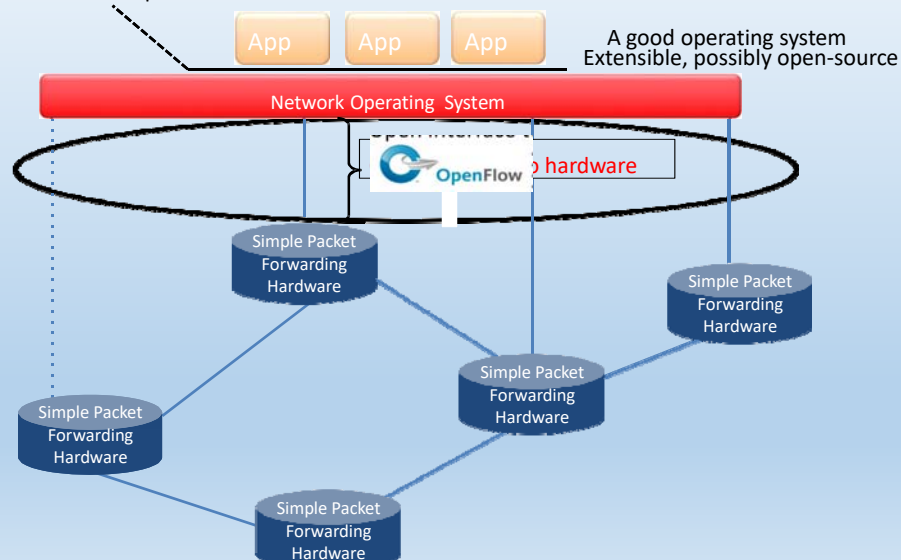
- Abstractions describe the core principles
- Interfaces are instantiations of abstractions
 - Provide concrete implementation specification, e.g., message or data format, memory layout, function call signature, ...
- Interfaces shield implementation details
 - Implementation freedom on both sides
 - Leads to modularity and exchangeability

A forwarding ~~abstraction~~ interface

- De-facto standard: OpenFlow

OpenFlow : A “southbound” API

Well-defined open API



A limitation on what OpenFlow is

- SDN is not OpenFlow
- OpenFlow is a part of SDN
 - “OpenFlow is one possible definition of how to model the configuration of a physical device” (S. Shenker)
 - OpenFlow is a protocol for remotely controlling the forwarding table of a switch or router
 - Essentially, it is the open interface to hardware
- OpenFlow
 - Is open source
 - Has cross-vendor support
 - Has academic and industrial backing

Two Important Aspects to OpenFlow

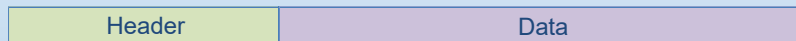
Two important design decisions for its success

1. Switches accept external control messages
 - Configuration comes from outside
2. Standardized flow entry format
 - So switches are interchangeable

OpenFlow Rules: <Match, Action>

- **Match**

- Match on any header, or new header
- Allows any flow granularity (edge vs core)



Match: 1000x01xx0101001x

- **Action**

- Forward to port(s), drop, send to controller
- Overwrite header with mask, push or pop
- Forward at specific bit-rate
- Allows multiple actions

OpenFlow properties (idealized)

- **Protocol independent**

- Construct Ethernet, IPv4, VLAN, MPLS, ...
- Construct new forwarding methods

- **Backward compatible**

- Run in existing networks

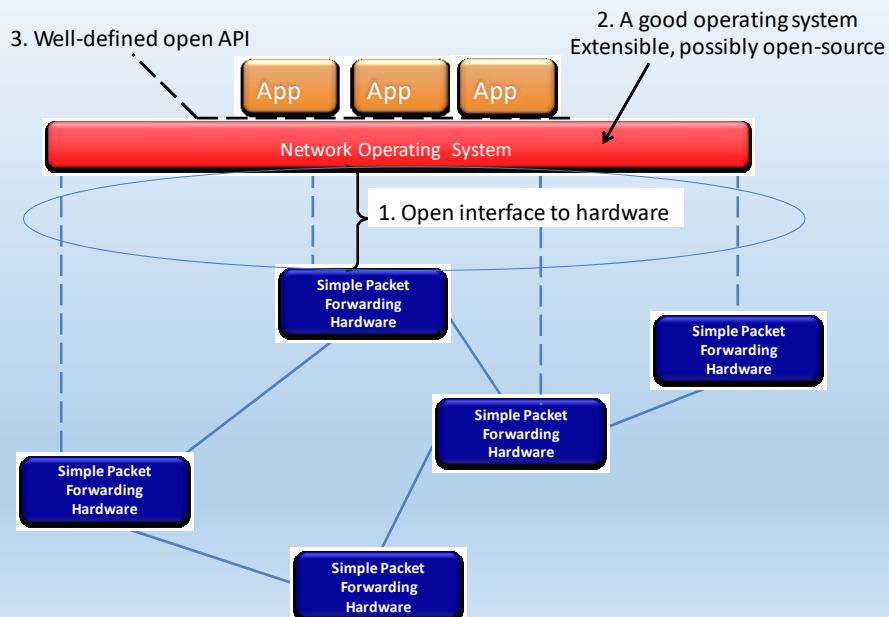
- **Technology independent**

- Switches, routers, WiFi APs
- Cellular base stations
- WDM/TDM circuits

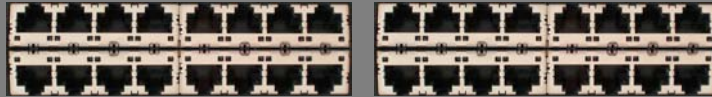


How does OpenFlow work?

The “Software-defined Network”



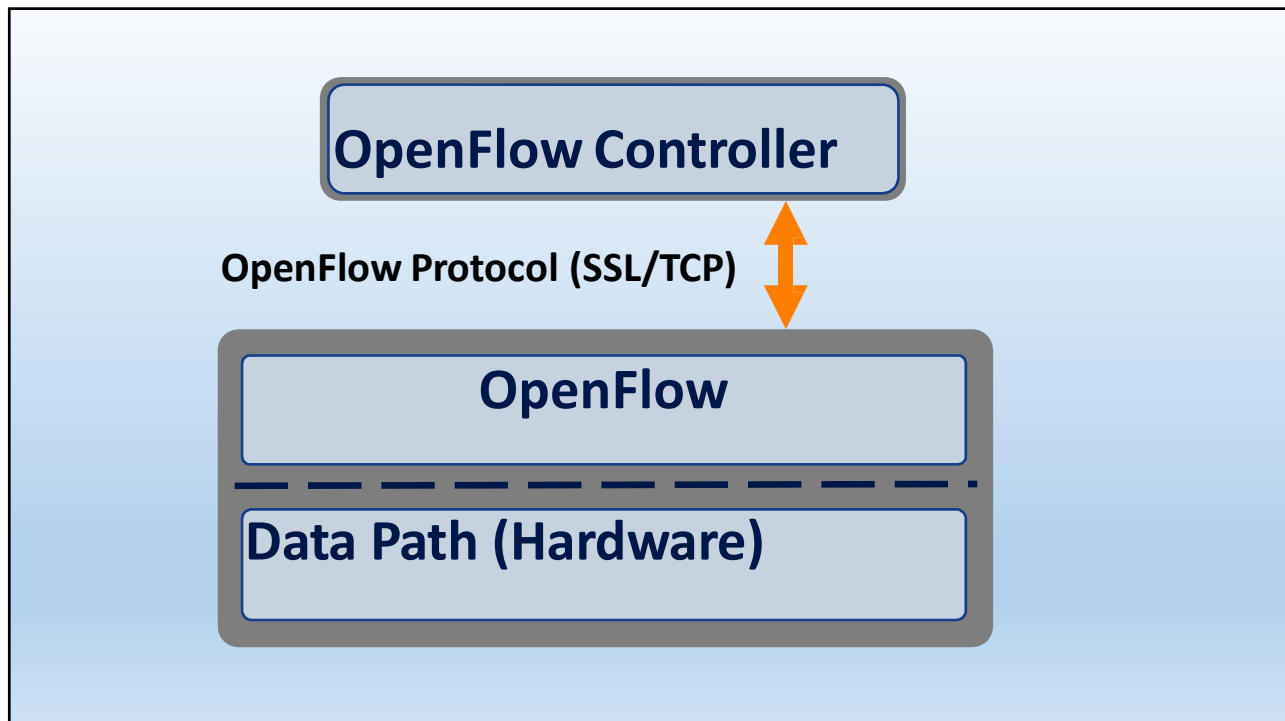
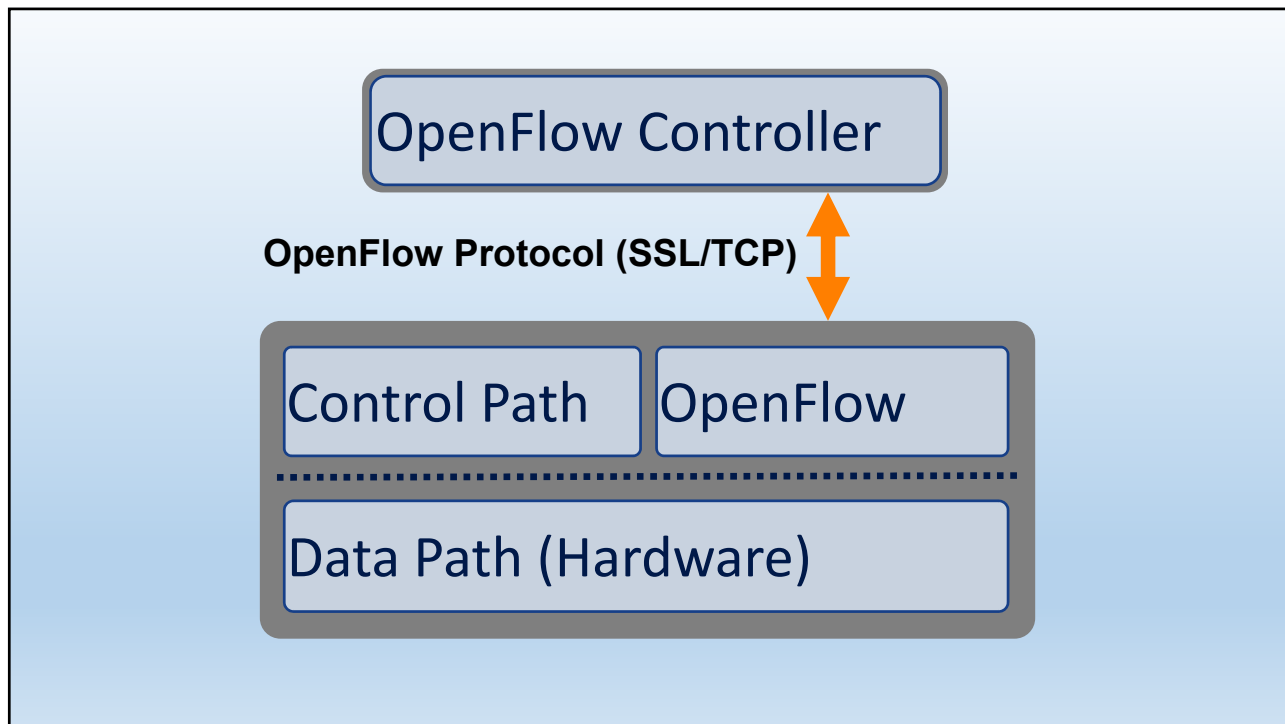
Ethernet Switch



Ethernet Switch

Control Path (Software)

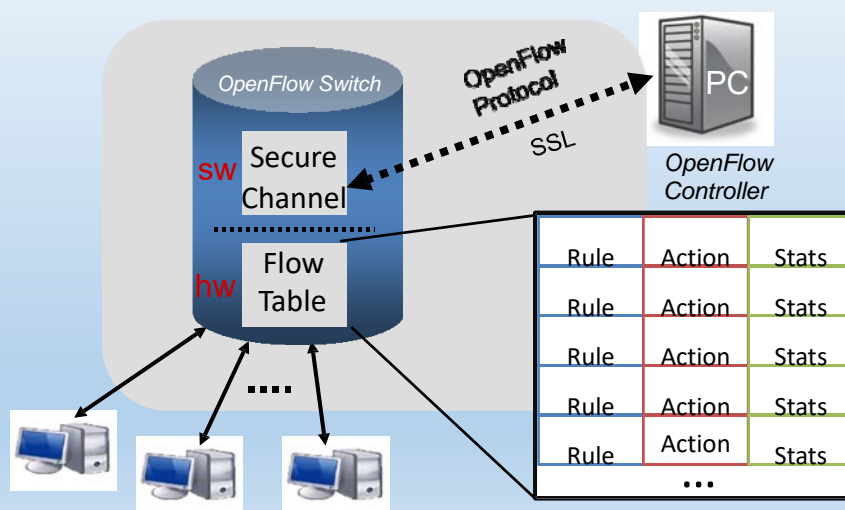
Data Path (Hardware)



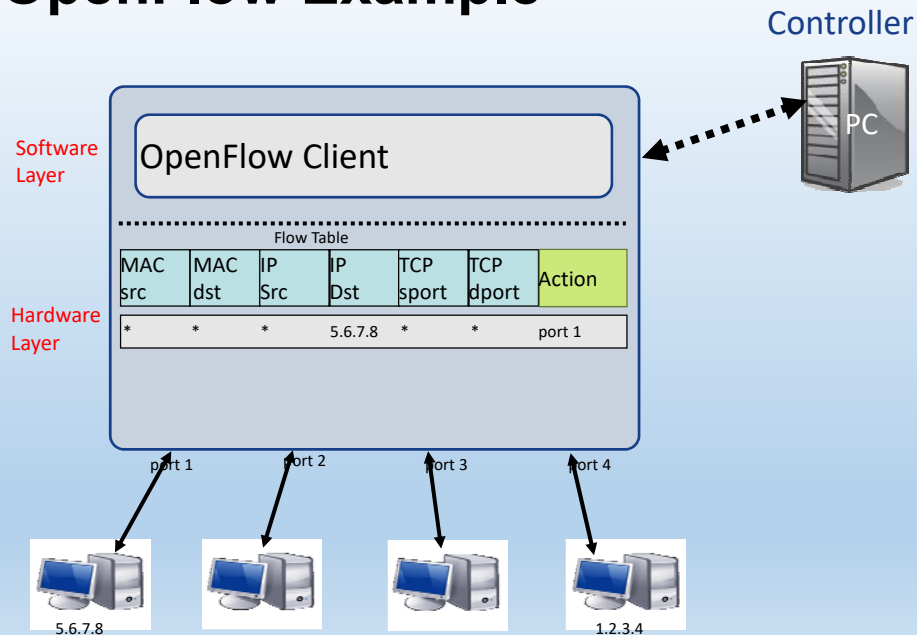
Secure Channel

- SSL Connection, site-specific key
- Controller discovery protocol
- Encapsulate packets for controller
- Send link/port state to controller
- Switch initiates connection to controller
 - Optional: Controller can also initiate connection

OpenFlow Architecture

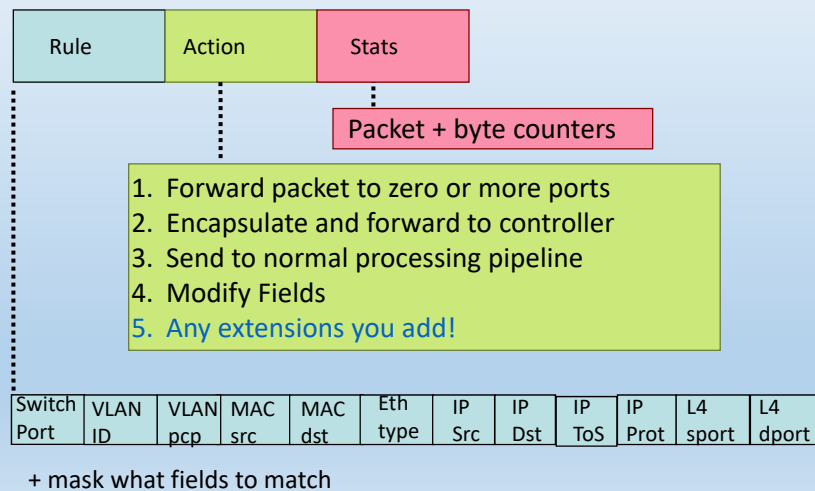


OpenFlow Example



OpenFlow Basics

Flow Table Entries



Flow Table Entries

Main components of a flow entry in a flow table

Match fields	To match against packets. These consist of the ingress port and packet headers
Priority	Matching precedence of the flow entry
Counters	e.g. packet and byte counters
Instructions	Determine action set or pipeline processing
Timeouts	Maximum amount of time or idle time before flow is expired by the switch
Cookies	Opaque data value chosen by the controller. Not used when processing packets.

Flow Table Entries

- Exact rules
 - All fields are specified
 - At most one rule per active flow
 - Higher priority than wildcard rules

ingress port	Eth dst	Eth src	Eth type	...	statistics	action
5	00:13:...	00:07:...	0x0800	...	counters	act 0

- Wildcard rules
 - At least one field contains a wildcard or a prefix
 - Multiple rules can match a packet => priorities

ingress port	Eth dst	Eth src	Eth type	...	statistics	action
*	00:10:...	*	*	...	counters	act 0

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

Examples

Routing

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	*	*	*	*	5.6.7.8	*	*	*	port6

VLAN Switching

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	00:1f:..	*	vlan1	*	*	*	*	*	port6, port7, port9

OpenFlow Actions

- **Forward** packet to physical / virtual port
 - **all**: broadcast to all ports (except incoming)
 - **controller**: send headers or entire packet to controller
 - **local**: send packet to local net
 - **table**: send to the next table (when pipelining)
 - **in_port**: send packet to input port
 - **normal**: forward using traditional Ethernet switch
 - **flood**: send along minimum spanning tree (except incoming port)
- **Enqueue** packet to a particular queue in the port (QoS)
- **Drop** packet
- **Modify** field: add/remove VLAN tags, ToS, TTL, IP address, port, etc.

OpenFlow packet handling

- If header matches an entry:
 - corresponding actions are performed and counters are updated
- If no header match:
 - Packet-In: packet is buffered and **packet header** is sent to the controller
 - Controller sends Packet-Out with action(s).
 - Optional: Controller sends a rule to handle packets in same flow
- Controller can send flow table entries proactively or reactively

Statistics in OpenFlow switches

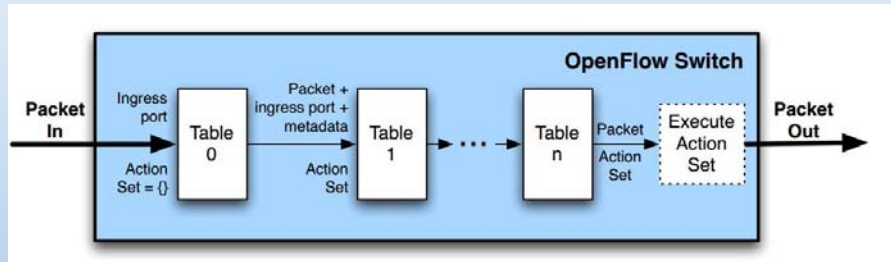
- Per Table
 - active entries, packet lookups, matches
- Per Flow
 - received packets, received bytes, duration
- Per Port
 - receive: packets, bytes, drops, errors, overrun errors
 - transmit: packets, bytes, drops, errors
 - collisions
- Per Queue
 - transmitted packets, bytes, overrun errors

Groups and meter table on switch

- **Groups** represent sets of actions for more complex forwarding semantics (e.g. flooding, multipath)
- A **meter table** consists of per-flow meters. A meter measures and enables controlling the rate of packets assigned to it and allows to implement simple QoS operations, e.g., rate limit packets to controller

OpenFlow Pipeline

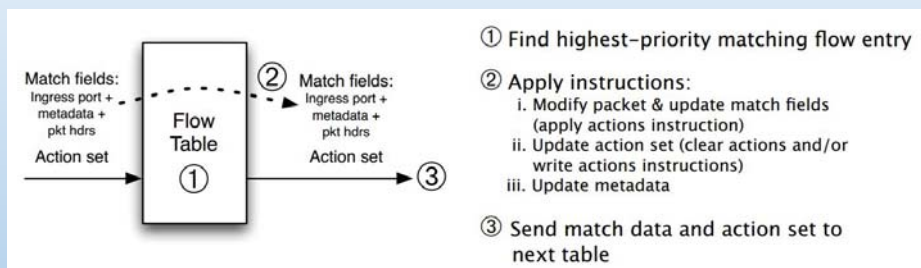
Packets are matched against multiple tables in the pipeline



OpenFlow Switch Specification Version > 1.1.0

OpenFlow Pipeline

Per-table packet processing



OpenFlow Switch Specification Version > 1.1.0

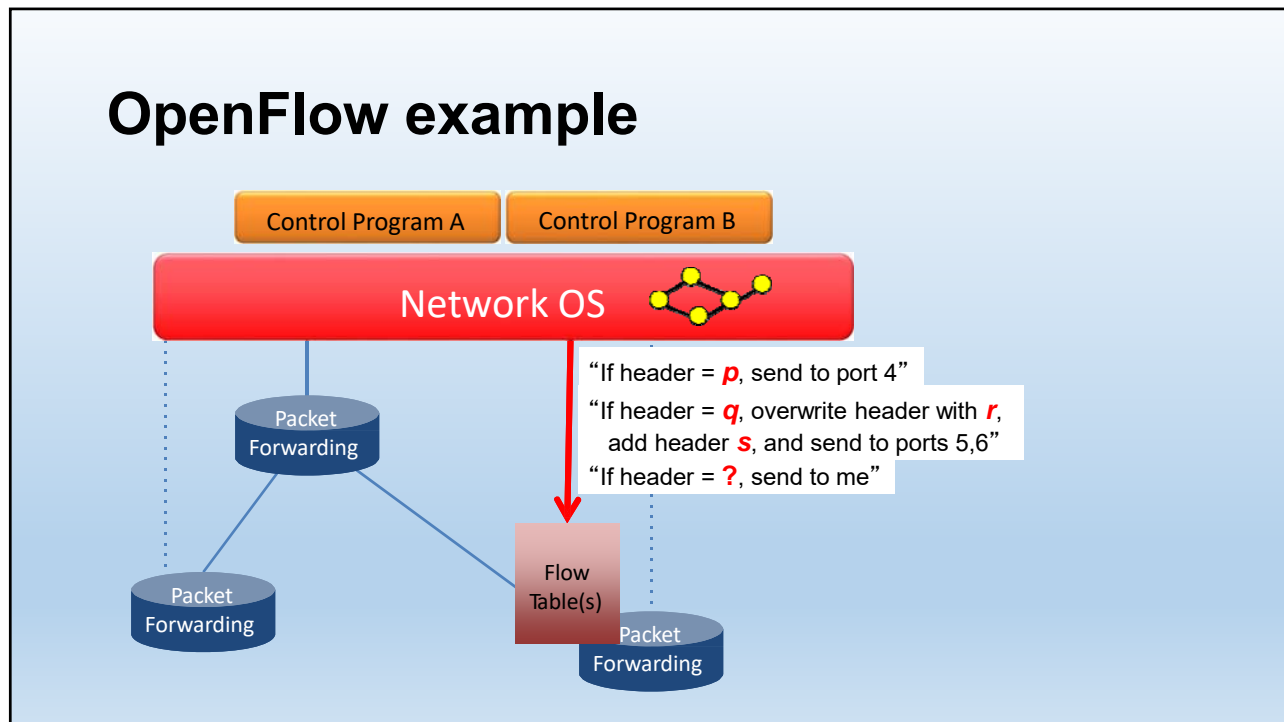
OpenFlow Message Types

- **Controller-to-switch messages**
 - manage flow entries
 - request info on switch capabilities and counters
 - send a packet back to a switch
- **Asynchronous messages**
 - send to controller a packet that does not match
 - inform controller that a timer has expired or that an error has occurred
- **Symmetric messages**
 - hello and echo messages

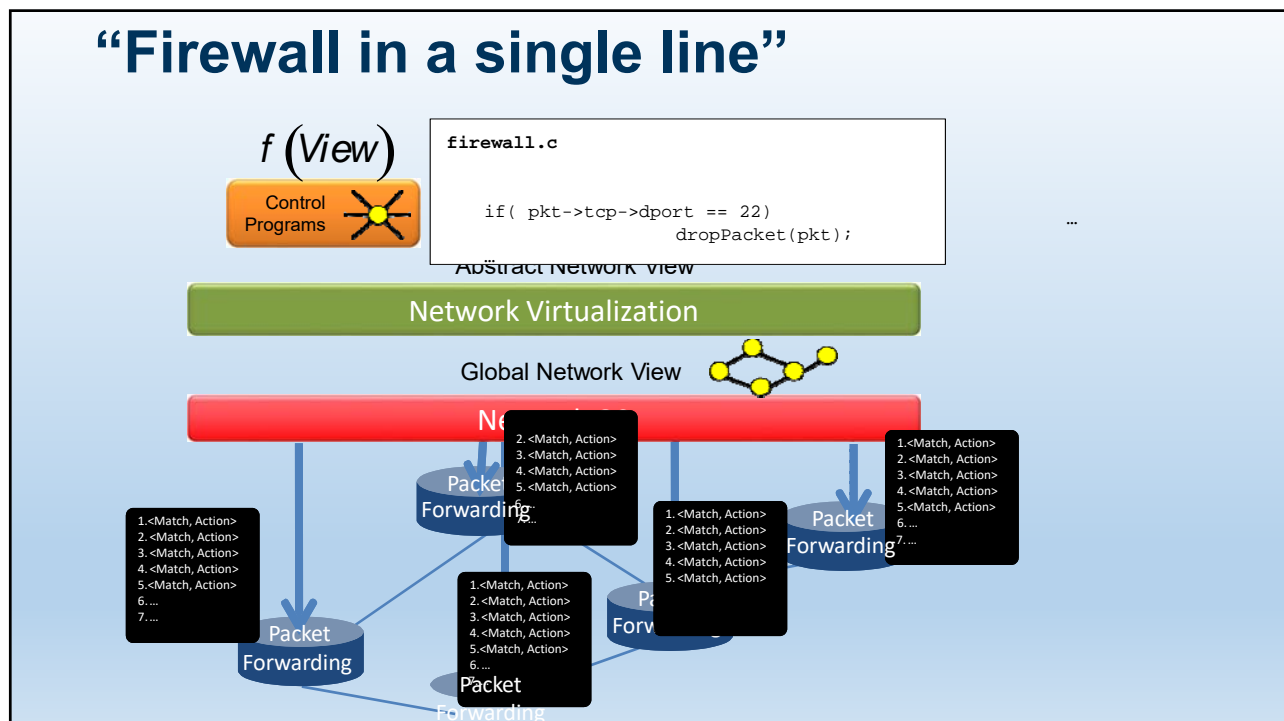
OpenFlow Key Messages

Message	Direction	Description
Packet-in	Switch → Controller	Transfer the control of a packet to the controller. Packet-in events can be configured to buffer packets
Packet-out	Controller → Switch	Instruct switch to send a packet out of a specified port. Sent in response to Packet-in messages
Modify-state	Controller → Switch	Add, delete and modify flow/group entries in the flow tables (aka Flow-mod); set switch port properties
Flow-removed	Switch → Controller	Inform the controller about the removal of a flow entry from a flow table

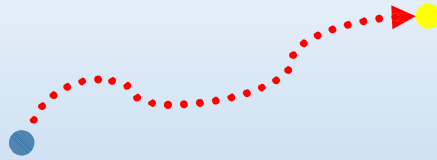
OpenFlow example



"Firewall in a single line"



Packets, routes, flows, flow entries, wait a minute...



What is a flow?

- Application flow
- All http
- Arshad's traffic
- All packets to Canada
- ← ...

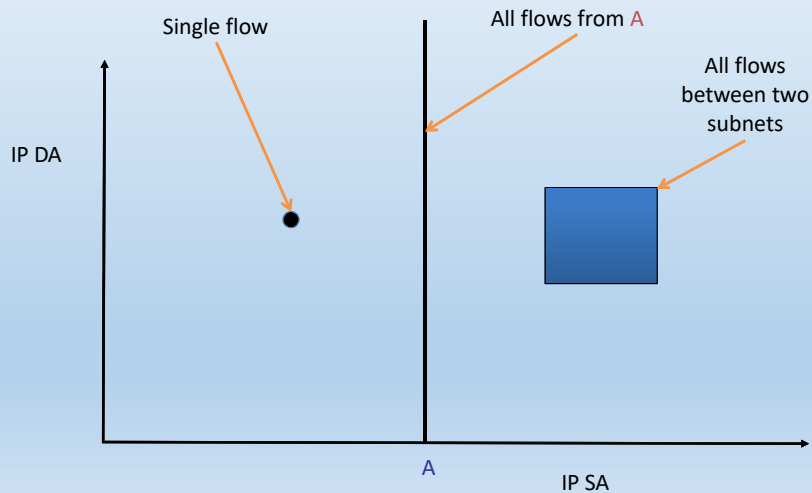
Types of action

- Allow/deny flow
- Route & re-route flow
- Isolate flow
- Remove flow

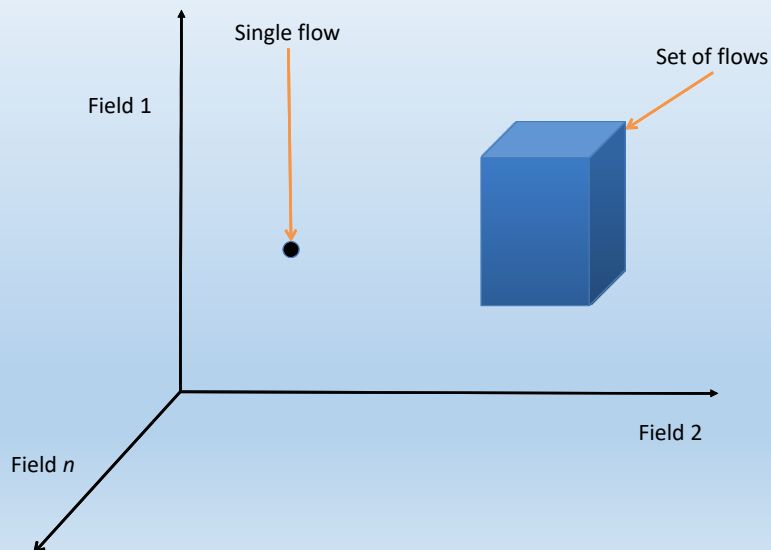
What do we need from the flow-based substrate?

- We need **flexible definitions of a flow**
 - Unicast, multicast, waypoints, load-balancing
 - Different aggregations
- We need **direct control over flows**
 - Flow as an entity we program: To route, to make private, to move, ...
- Exploit the **benefits of packet switching**
 - It works and is universally deployed
 - It's efficient (when kept simple)

Flowspace: A way to think about flows defined by match fields



Flowspace: Generalization

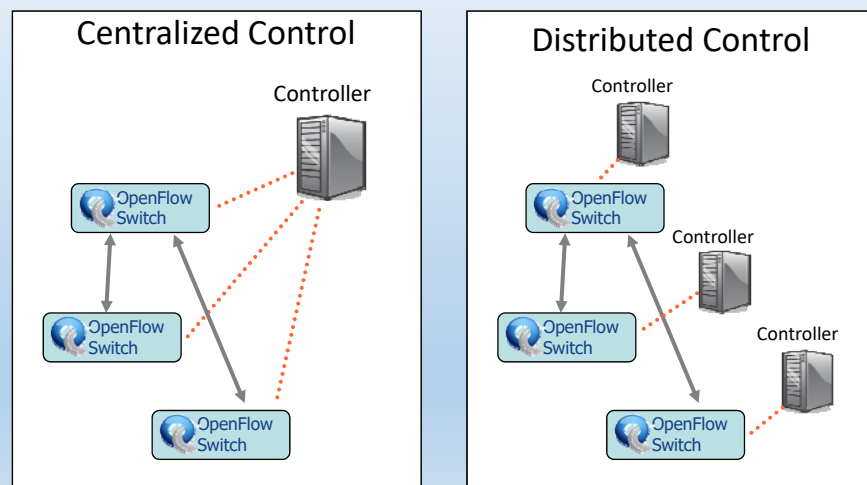


Properties of FlowSpace

- Backwards compatible
 - Current layers are a special case
 - No end points need to change
- Easily implemented in hardware
 - e.g. TCAM flow-table in each switch
- Strong isolation of flows
 - Simple geometric construction
 - Can prove which flows can/cannot communicate

Centralized vs Distributed Control

(Design Decision) Both models are possible with OpenFlow



Flow Routing vs. Aggregation (Design Decision)

Both models are possible with OpenFlow

Flow-Based

- Every flow is individually set up by controller
- Exact-match flow entries
- Flow table contains one entry per flow
- Good for fine grain control, e.g. campus networks

Aggregated

- One flow entry covers large groups of flows
- Wildcard flow entries
- Flow table contains one entry per category of flows
- Good for large number of flows, e.g. backbone

Reactive vs. Proactive (pre-populated)

(Design Decision) Both models are possible with OpenFlow

Reactive

- First packet of flow triggers controller to insert flow entries
- Efficient use of flow table
- Every flow incurs small additional flow setup time
- If control connection lost, switch has limited utility

Proactive

- Controller pre-populates flow table in switch
- Zero additional flow setup time
- Loss of control connection does not disrupt traffic
- Essentially requires aggregated (wildcard) rules

OpenFlow's Evolution Path

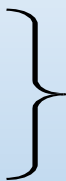
- [OF v1.0](#) (end of 2009): Single table, L2+IPv4 focused matching
 - Most widely used version
- [OF v1.1](#) (Mar 2011):
 - multiple tables, MPLS + VLAN matching, multipath forwarding: ECMP, groups
- [OF v1.2](#) (Dec 2011): "Extensible Protocol"
 - extensible match & actions (Type/Length/Value), IPv6, multiple controllers
- [OF Config. & Man. 1.0](#) (Dec 2011): Basic configuration: queues, ports, controller assignment
- [OF v1.3](#) (June 2012): Better expression of capabilities of a switch, tunnels, meters, multiple parallel channels between switch and controller, more on IPv6
- [OF Conformance Test](#) (2012): Interoperability and conformance test process for OpenFlow switches
- [OF v1.4](#) (Aug 13): Improve extensibility, better support for optical ports, many other incremental improvements
- [OF v1.5.1](#) (May 15): Egress Tables, Packet type aware pipeline , many other incremental improvements
- **Many more intermediate releases with errata and smaller improvements**

OpenFlow Support

- [Open Networking Foundation](#) was founded in 2011 to develop and standardize OpenFlow
 - Members include Cisco, Facebook, Google, HP, IBM, Juniper Networks etc.
- Juniper and start-ups Nicira and Big Switch are warm supporters of OpenFlow
- Vendors, such as the Cisco, IBM, NEC and HP have implemented OpenFlow in existing products
- Cisco's SDN initiative is called Open Network Environment (ONE)
- **Unusual to see such speedy collaboration between academia and industry**

OpenFlow Usage Models

1. Experiments at the **flow level**

- User-defined routing protocols
 - Admission control
 - Network access control
 - Network management
 - Energy management
 - VOIP mobility and handoff
- 
- Experiment-specific controllers
 - Static or dynamic flow-entries

2. Experiments at the **packet level**

- Slow: Controller handles packet processing
- Fast: Redirect flows through programmable hardware
- Modified routers, firewalls, NAT, congestion control...

OpenFlow: Switches Support




Switches– Commercial

1. Arista
2. Brocade MLX/NetIron Products
3. Extreme BlackDiamond X8
4. Huawei
5. HP ProCurve
6. IBM BNT G8264
7. Juniper MX Series (SDK)
8. NEC
9. NetGear
10. Pronto
11. Many more smaller vendors

Switches – Open Source

1. Open vSwitch
2. NetFPGA reference implementation
3. OpenWRT/Pantou
4. Mininet (emulation)
5. Many more

Commercial Switch Vendors

Model	Virtualize	Notes	
HP Procurve 5400zl or 6600	1 OF instance per VLAN	<ul style="list-style-type: none"> -LACP, VLAN and STP processing before OpenFlow -Wildcard rules or non-IP pkts processed in s/w -Header rewriting in s/w -CPU protects mgmt during loop 	
NEC IP8800	1 OF instance per VLAN	<ul style="list-style-type: none"> -OpenFlow takes precedence -Most actions processed in hardware -MAC header rewriting in h/w 	
Pronto 3240 or 3290 with Pica8 or Indigo firmware	1 OF instance per switch	<ul style="list-style-type: none"> -No legacy protocols (like VLAN and STP) -Most actions processed in hardware -MAC header rewriting in h/w 	

89

Current SDN hardware

Juniper MX-series



NEC IP8800



WiMax (NEC)



HP Procurve 5400



Netgear 7324



PC Engines



Pronto 3240/3290



Ciena Coredirector



More coming
soon...

OpenFlow: Controllers Support

- **Controllers – Commercial**
 - Big Switch Networks
 - NEC ProgrammableFlow Controller
 - Nicira NVP
- **Controllers – Open Source**
 - NOX (C++/Python)
 - Beacon (Java)
 - Floodlight (Java)
 - OpenDayLight (Java)
 - Maestro (Java)
 - RouteFlow (NOX, Quagga, ...)

What can you not do with OpenFlow ver1.0

- Non-flow-based (per-packet) networking
 - ex. Per-packet next-hop selection (in wireless mesh)
 - yes, this is a fundamental limitation
 - **BUT OpenFlow can provide the plumbing to connect these systems**
- Use all tables on switch chips
 - yes, a major limitation (cross-product issue)
 - **BUT OpenFlow 1.3 version will expose these**

What can you not do with OpenFlow ver1.0

- New forwarding primitives
 - BUT provides a nice way to integrate them through extensions
- New packet formats/field definitions
 - BUT a generalized OpenFlow (2.0) is on the horizon
- Optical Circuits
 - BUT efforts underway to apply OpenFlow model to circuits
- Low-setup-time of individual flows
 - BUT can push down flows proactively to avoid delays

Where it's going

- OF v1.5
 - multiple tables: leverage additional tables
 - tags and tunnels
 - multipath forwarding
 - per flow meters
- OF v2+
 - generalized matching and actions: protocol independent forwarding

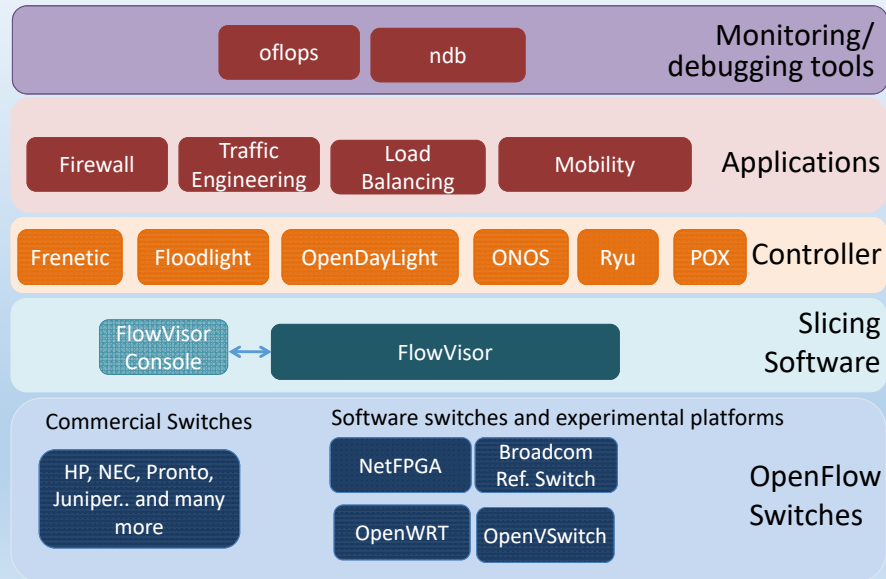
Outline

- Review of previous lecture
- SDN Basics
 - Concepts
 - OpenFlow
 - **Switches and Controllers**
 - OF-Config
 - Mininet

Switches and Controllers

- OpenFlow switches and vendors
- Controllers
 - Floodlight

OpenFlow building blocks



Controller Vendors

Vendor	Notes
Nicira's NOX	<ul style="list-style-type: none"> •Open-source GPL •C++ and Python •Researcher friendly
Nicira's ONIX	<ul style="list-style-type: none"> •Closed-source •Datacenter networks
Ryu	<ul style="list-style-type: none"> •Open-source GPL •Python

Vendor	Notes
Stanford's Beacon	<ul style="list-style-type: none"> •Open-source •Researcher friendly •Java-based
BigSwitch controller	<ul style="list-style-type: none"> •Ha open source version •Based on Beacon •Enterprise network
OpenDayLight	<ul style="list-style-type: none"> •Open-source •Based on Java
Frenetic	<ul style="list-style-type: none"> •Open-source •Written in functional programming languages

Floodlight Architecture **Overview**



- Floodlight is a collection of modules
- Some modules (not all) export services
- All modules in Java
- Rich, extensible REST API

Floodlight Architecture **Module descriptions**



- Translates OF messages to Floodlight events
- Managing connections to switches via Netty
- Computes shortest path using Dijkstra
- Keeps switch to cluster mappings
- Maintains state of links in network
- Sends out LLDPs
- Installs flow mods for end-to-end routing
- Handles island routing
- Tracks hosts on the network
- MAC -> switch,port, MAC->IP, IP->MAC
- DB style storage (queries, etc)
- Modules can access all data and subscribe to changes
- Implements via Restlets (restlet.org)
- Modules export RestletRoutable
- Supports the insertion and removal of static flows
- REST-based API
- Create layer 2 domain defined by MAC address
- Used for OpenStack / Quantum

10
100
0

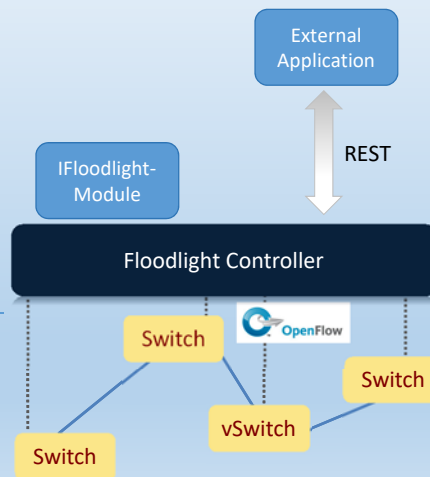
Floodlight Programming Model Northbound APIs

IFloodlightModule

- Java module that runs as part of Floodlight
- Consumes services and events exported by other modules
 - OpenFlow (ie. Packet-in)
 - Switch add / remove
 - Device add /remove / move
 - Link discovery

External Application

- Communicates with Floodlight via REST
 - Quantum / Virtual networks
 - Normalized network state
 - Static flows



REST API Reference

A moving target...but...

