Towards Software-Defined Networks

Network Infrastructures

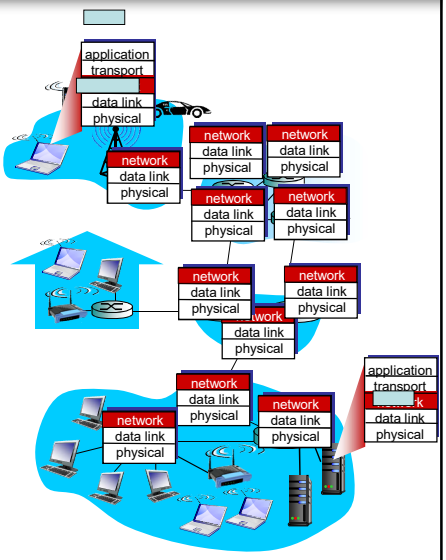
Based on Slides from Nick McKewon, Scott Shenker, Kurose-Ross, Tim Hinrichs and Tommaso Melodia

Outline

- A brief review: How do current networks work
- Software-defined Networking Paradigm
- Network Virtualization and OpenFlow
- Flowvisor
- Network operating systems
- Debugging through software-defined networks

Network Layer

- transport segment from sending to receiving host
- on sending side encapsulates segments into datagrams
- on receiving side, delivers segments to transport layer
- network layer protocols in *every* host, router
- router examines header fields in all IP datagrams passing through it



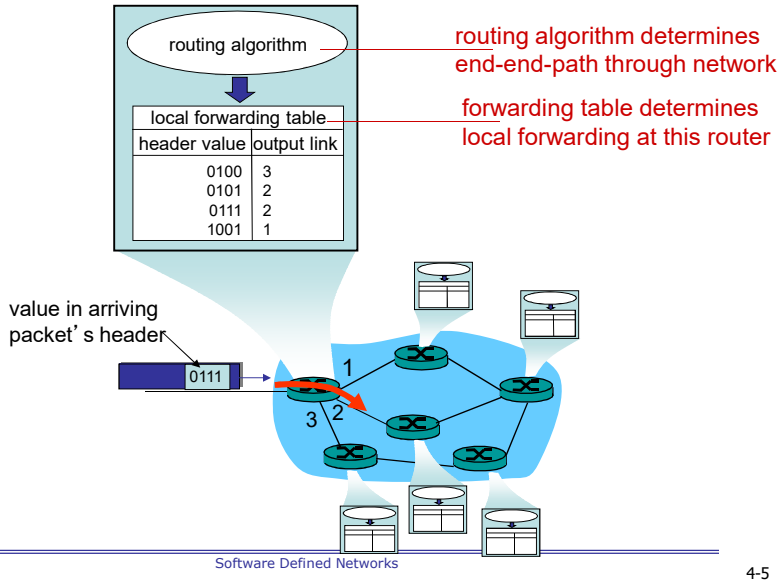
Two Key Network-layer Functions

- *forwarding:* move packets from router' s input to appropriate router output
- *routing:* determine route taken by packets from source to dest.
  - *routing algorithms*

analogy:

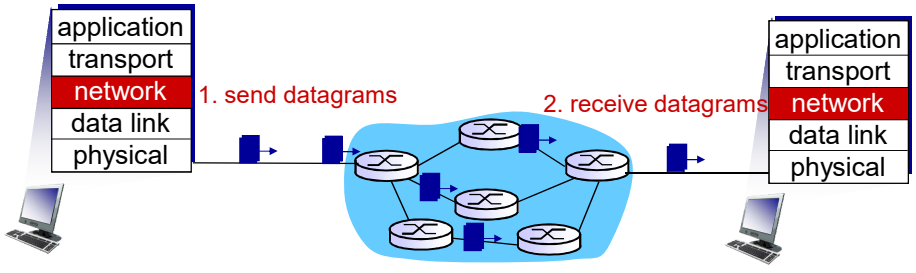
- *routing:* process of planning trip from source to dest
- *forwarding:* process of getting through single interchange

Interplay Between Routing and Forwarding

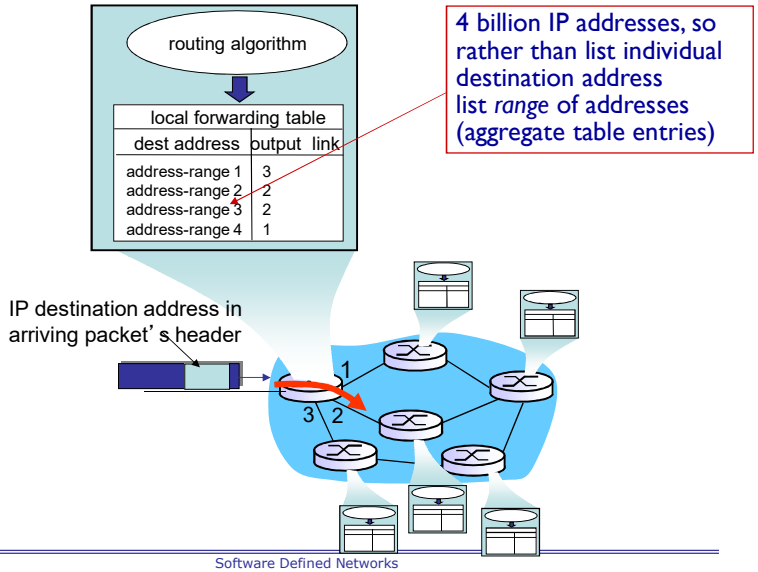


Datagram Networks

- no call setup at network layer
- routers: no state about end-to-end connections
  - no network-level concept of "connection"
- packets forwarded using destination host address



Datagram Forwarding Table



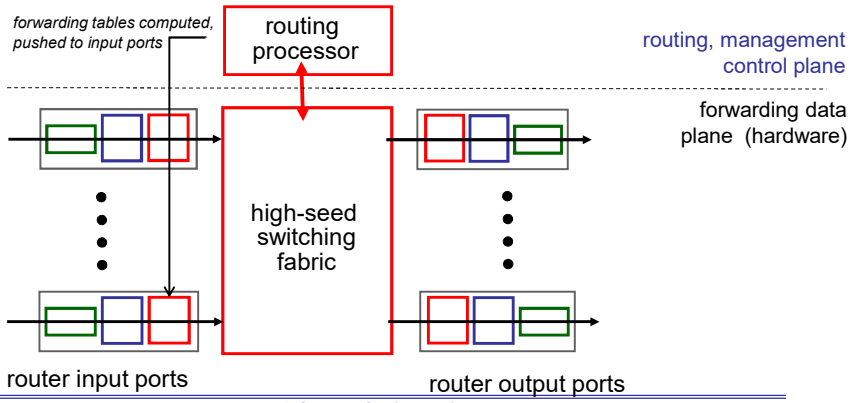
Datagram Forwarding Table

Destination Address Range	Link Interface
11001000 00010111 00010000 00000000 through 11001000 00010111 00010111 11111111	0
11001000 00010111 00011000 00000000 through 11001000 00010111 00011000 11111111	1
11001000 00010111 00011001 00000000 through 11001000 00010111 00011111 11111111	2
otherwise	3

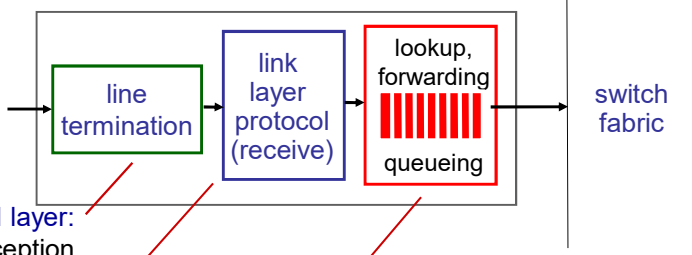
Router Architecture Overview

two key router functions:

- run routing algorithms/protocol (RIP, OSPF, BGP)
- *forwarding* datagrams from incoming to outgoing link



Input Port Functions



physical layer:
bit-level reception

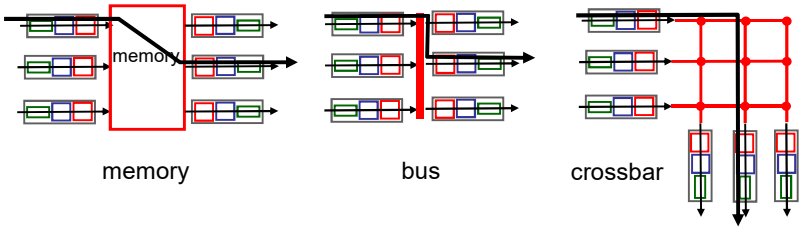
data link layer:
e.g., Ethernet

decentralized switching:

- given datagram dest., lookup output port using forwarding table in input port memory *("match plus action")*
- goal: complete input port processing at 'line speed'
- queuing: if datagrams arrive faster than forwarding rate into switch fabric

Switching Fabrics

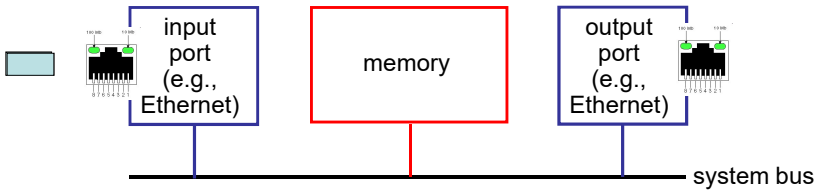
- transfer packet from input buffer to appropriate output buffer
- switching rate: rate at which packets can be transferred from inputs to outputs
  - often measured as multiple of input/output line rate
  - N inputs: switching rate N times line rate desirable
- three types of switching fabrics



Switching Via Memory

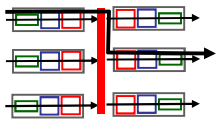
first generation routers:

- traditional computers with switching under direct control of CPU
- packet copied to system's memory
- speed limited by memory bandwidth (2 bus crossings per datagram)



Switching Via a Bus

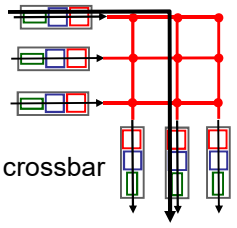
- datagram from input port memory to output port memory via a shared bus
- *bus contention:* switching speed limited by bus bandwidth
- 32 Gbps bus, Cisco 5600: sufficient speed for access and enterprise routers



bus

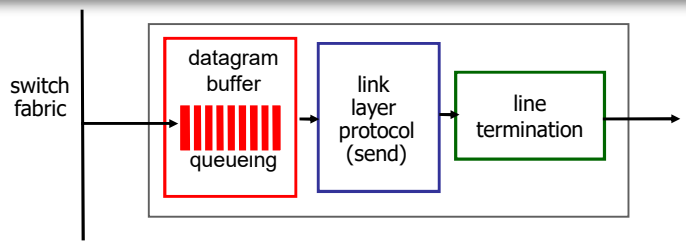
Switching Via Interconnection Network

- overcome bus bandwidth limitations
- banyan networks, crossbar, other interconnection nets initially developed to connect processors in multiprocessor
- advanced design: fragmenting datagram into fixed length cells, switch cells through the fabric.
- Cisco 12000: switches 60 Gbps through the interconnection network



crossbar

Output Ports

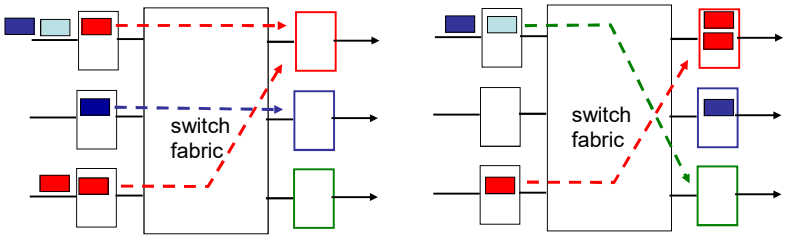


- *buffering* required when datagrams arrive from fabric faster than the transmission rate
- *scheduling discipline* chooses among queued datagrams for transmission

Datagram (packets) can be lost due to congestion, lack of buffers

Priority scheduling – who gets best performance, network neutrality

Output Port Queueing



at *t*, packets more from input to output

one packet time later

- buffering when arrival rate via switch exceeds output line speed
- *queueing (delay) and loss due to output port buffer overflow!*

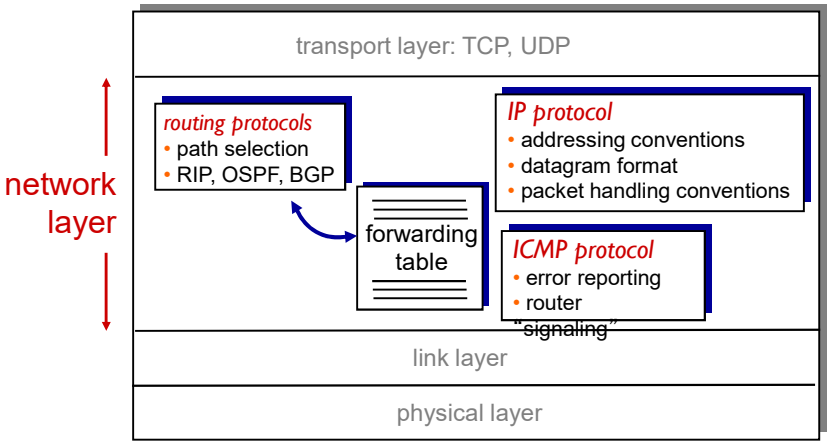
How much buffering?

- RFC 3439 rule of thumb: average buffering equal to "typical" RTT (say 250 msec) times link capacity C
  - e.g., C = 10 Gpbs link: 2.5 Gbit buffer
- recent recommendation: with *N* flows, buffering equal to

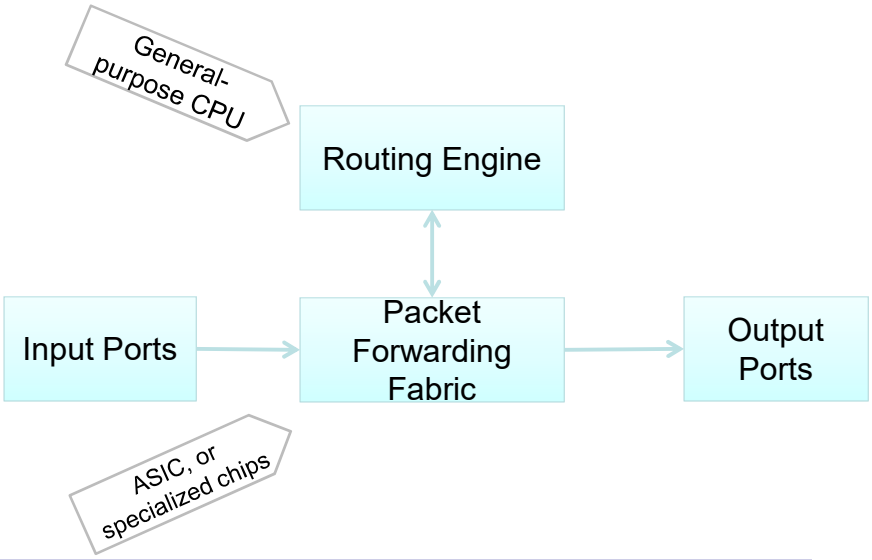
$$\frac{RTT \cdot C}{\sqrt{N}}$$

The Internet Network Layer

host, router network layer functions:



Inside a Router



Two Key Definitions

- **Data Plane**: processing and delivery of packets
  - Based on state in routers and endpoints
  - E.g., IP, TCP, Ethernet, etc.
  - Fast timescales (per-packet)
- **Control Plane**: establishing the state in routers
  - Determines how and where packets are forwarded
  - Routing, traffic engineering, firewall state, …
  - Slow time-scales (per control event)

We have lost our way

Routing, management, mobility management, access control, VPNs, …

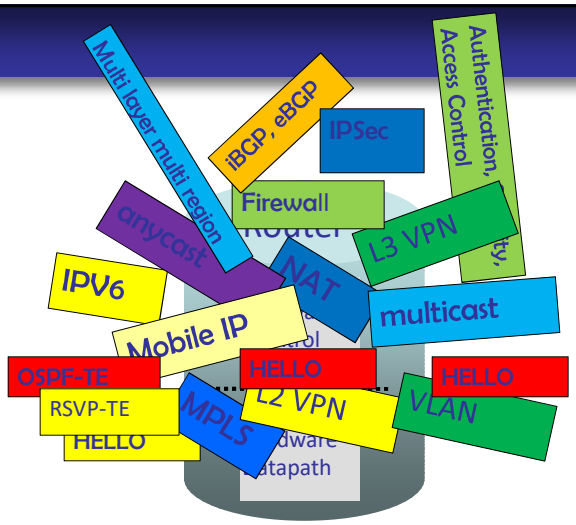
App | App | App

Operating System

Million of lines of source code | 5400 RFCs | Barrier to entry

Specialized Packet Forwarding Hardware

500M gates 10Gbytes RAM | Bloated | Power Hungry



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"

Software Defined Networks: The Future of Networking and the Past of Protocols

Key to Internet Success: Layers

Applications

…built on…

Reliable (or unreliable) transport

…built on…

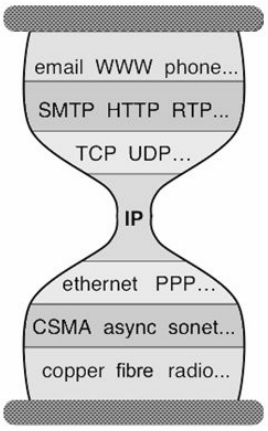
Best-effort global packet delivery

…built on…

Best-effort local packet delivery

…built on…

Physical transfer of bits



Why Is Layering So Important?

- Decomposed delivery into fundamental components
- Independent but compatible **innovation** at each layer
- A practical success of unprecedented proportions…
- **…but an academic failure**

Built an Artifact, Not a Discipline

- Other fields in "systems": Operating Systems, Data Bases, Distributed Systems
  - Teach basic principles
  - Are easily managed
  - Continue to evolve
- Networking:
  - Teach big bag of protocols
  - Notoriously difficult to manage
  - Evolves very slowly

Why Does Networking Lag Behind?

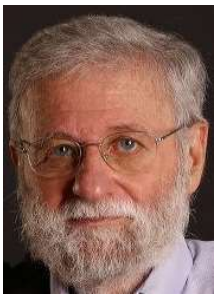
- Networks used to be simple: Ethernet, IP, TCP….
- New **control** requirements led to great complexity
  - Isolation ➔ VLANs, ACLs
  - Traffic engineering ➔ MPLS, ECMP, Weights
  - Packet processing ➔ Firewalls, NATs, middleboxes
  - Payload analysis ➔ Deep packet inspection (DPI)
  - …..
- Mechanisms designed and deployed independently
  - Complicated "control plane" design, primitive functionality
  - Stark contrast to the elegantly modular "data plane"

Infrastructure Still Works!

- **Only** because of "our" ability to master complexity
- This ability to master complexity is both a blessing…
  - **…and a curse!**

A Simple Story About Complexity

- ~1985: Don Norman visits Xerox PARC
  - Talks about user interfaces and stick shifts



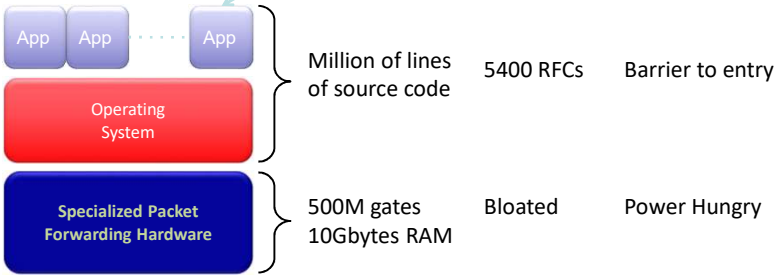
What Was His Point?

- The ability to ***master complexity*** is not the same as the ability to ***extract simplicity***
- When first getting systems to work….
  - Focus on mastering complexity
- When making system easy to use and understand
  - Focus on extracting simplicity
- **You will never succeed in extracting simplicity**
  - If don't recognize it is different from mastering complexity

Take Home Message

- Networking still focused on mastering complexity
  - Little emphasis on extracting simplicity from control plane
  - No recognition that there is a difference….
- Extracting simplicity builds intellectual foundations
  - **Necessary for creating a discipline….**
  - **That's why networking lags behind**

We have lost our way

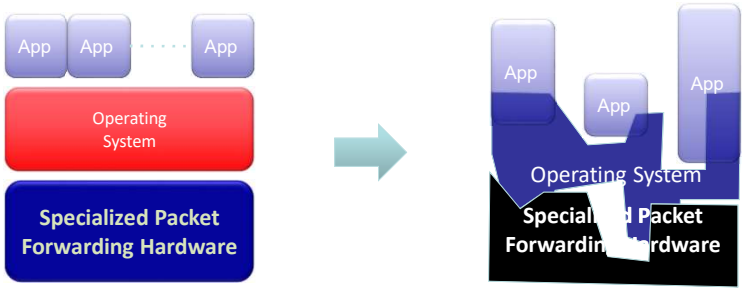


Many complex functions baked into the infrastructure

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

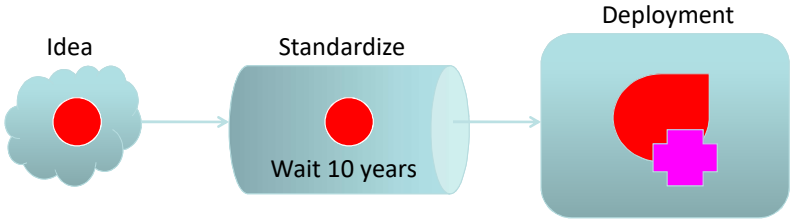
An industry with a "mainframe-mentality"

Reality



- Lack of competition means glacial innovation
- Closed architecture means blurry, closed interfaces

Glacial process of innovation made worse by captive standards process



- Driven by vendors
- Consumers largely locked out
- Lowest common denominator features
- Glacial innovation

A Better Example: Programming

- Machine languages: no abstractions
  - Mastering complexity was crucial
- Higher-level languages: OS and other abstractions
  - File system, virtual memory, abstract data types, ...
- Modern languages: even more abstractions
  - Object orientation, garbage collection,…

Abstractions key to extracting simplicity

Why Are Abstractions/Interfaces Useful?

- Interfaces are instantiations of abstractions
- Interfaces shield programs from low-level details
  - Allows freedom of implementation on both sides
  - Which leads to modular program structure
- They don't remove complexity, merely hide it
  - Someone deals with complexity once
  - Everyone else leverages that work

"The Power of Abstraction"

"Modularity based on abstraction is the way things get done"

Barbara Liskov

Abstractions ➔ Interfaces ➔ Modularity

What abstractions do we have in networking?

Layers are Great Abstractions

- Layers only deal with the **data plane**
  - **IP**'s best effort delivery
  - **TCP**'s reliable byte-stream
- We have no powerful ***control plane*** abstractions!
  - No sophisticated management/control building blocks
  - So new control requirements cause increased complexity
- How do we find those control plane abstractions?
- Two steps: ***define*** problem, and then ***decompose*** it

The Network Control Problem

- Compute the configuration of each physical device
  - E.g., Forwarding tables, ACLs,…
- Operate without communication guarantees
- Operate within given network-level protocol

Only people who love complexity would find this a reasonable request

Programming Analogy

- What if programmers had to:
  - Specify where each bit was stored
  - Explicitly deal with all internal communication errors
  - Within a programming language with limited expressability
- Programmers would redefine problem:
  - Define a higher level abstraction for memory
  - Build on reliable communication abstractions
  - Use a more general language
- **Abstractions** divide problem into tractable pieces
  - And make programmer's task easier

From Requirements to Abstractions

1. Operate without communication guarantees
   Need an abstraction for **distributed state**
   - the communication in control programs is ALL directed towards collecting/disseminating/or calculating on distributed state
2. Compute the configuration of each physical device
   Need an abstraction that **simplifies configuration**
3. Operate within given network-level protocol
   Need an abstraction for general **forwarding model**

Once these abstractions are in place, control mechanism has a much easier job!

SDN in one sentence

- 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 create a discipline
  - Abstractions enable reasoning about system behavior
  - Provides environment where formalism can take hold….
- OK, but what are these abstractions?

1. Distributed State Abstraction

- Shield control mechanisms from state distribution
  - While allowing access to this state
- Natural abstraction: ***global network view***
  - Annotated network graph provided through an API
- Implemented with "Network Operating System"
- Control mechanism is now program using API
  - No longer a distributed protocol, now just a graph algorithm
  - E.g. Use Dijkstra rather than Bellman-Ford

Software Defined Network (SDN)

Network Control Platform for Routers

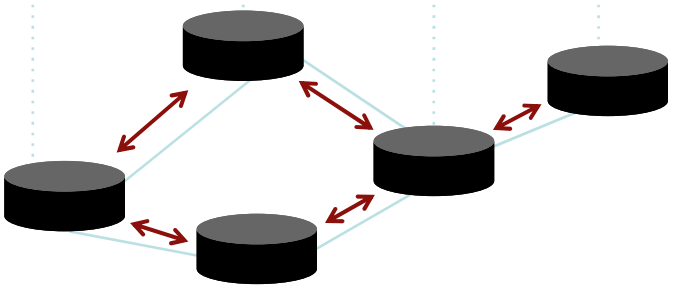
e.g. routing, access control

Control Program

Global Network View

Distributed algorithm running between neighbors

Network OS



Major Change in Paradigm

- No longer designing distributed control protocols
  - Design one distributed system (NOS)
  - Use for all control functions
- Now just defining a centralized control ***function***

Configuration = Function(view)

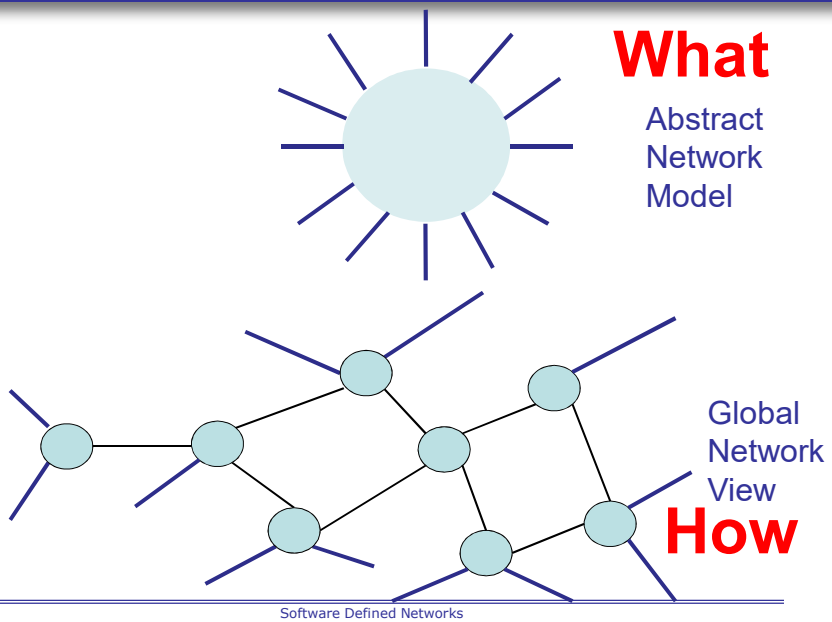
Key Task of Network Controller

- OpenFlow protocol is largely deltas:
  - Switch-to-Controller: changes of network state
  - Controller-to-Switch: changes of configuration
- It is a natural way to write control logic

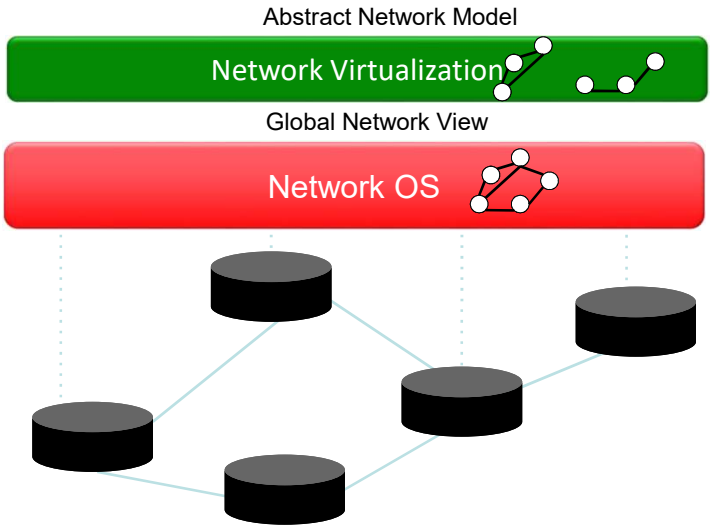
2. Specification Abstraction

- Control program should express desired behavior
- It should not be responsible for implementing that behavior on physical network infrastructure
- Natural abstraction: **simplified model** of network
  - Simple model with only enough detail to specify goals
- Requires a new shared control layer:
  - **Map abstract configuration to physical configuration**
- This is "network virtualization"

Simple Example: Access Control



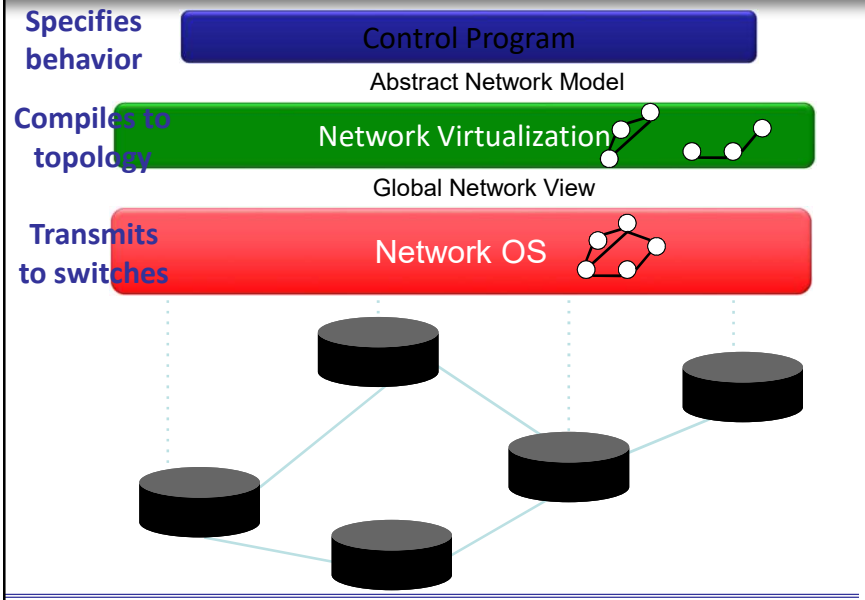
Software Defined Network: Take 2



What Does This Picture Mean?

- Write a simple program to configure a simple model
  - Configuration merely a way to specify what you want
- Examples
  - Access Control Lists: who can talk to who
  - Isolation: who can hear my broadcasts
  - Routing: only specify routing to the degree you care
    - Some flows over satellite, others over landline
  - Traffic Engineering: specify in terms of quality of service, not routes
- Virtualization layer "compiles" these requirements
  - Produces suitable configuration of actual network devices
- Network Operating System then transmits these settings to physical boxes

Software Defined Network: Take 2



Two Examples Uses

- Scale-out router:
  - Abstract view is single router
  - Physical network is collection of interconnected switches
  - Allows routers to "scale out, not up"
  - Use standard routing protocols on top
- Multi-tenant networks:
  - Each tenant has control over their "private" network
  - Network virtualization layer compiles all of these individual control requests into a single physical configuration
- **Hard to do without SDN, easy** *(in principle)* **with SDN**

3. Forwarding Abstraction

- Switches have two “brains”
  - Management CPU (smart but slow)
  - Forwarding ASIC (fast but dumb)
- Need a forwarding abstraction for both
  - CPU abstraction can be almost anything
- ASIC abstraction is much more subtle: **OpenFlow**
- OpenFlow:
  - Control switch by inserting <header;action> entries
  - Essentially gives NOS remote access to forwarding table
  - Instantiated in OpenvSwitch

Does SDN Work?

- Is it scalable? **Yes**
- Is it less responsive? **No**
- Does it create a single point of failure? **No**
- Is it inherently less secure? **No**
- Is it incrementally deployable? **Yes**

SDN: Clean Separation of Concerns

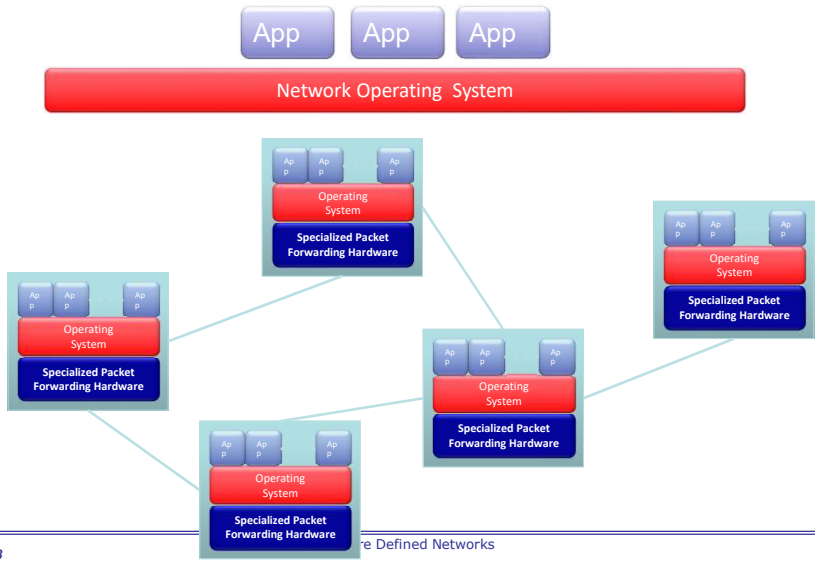
- **Control program: specify behavior on abstract model**
  - Driven by **Operator Requirements**
- **Network Virtualization: map abstract model to global view**
  - Driven by **Specification Abstraction**
- **Network Operating System: map global view to physical switches**
  - API: driven by **Distributed State Abstraction**
  - Switch/fabric interface: driven by **Forwarding Abstraction**

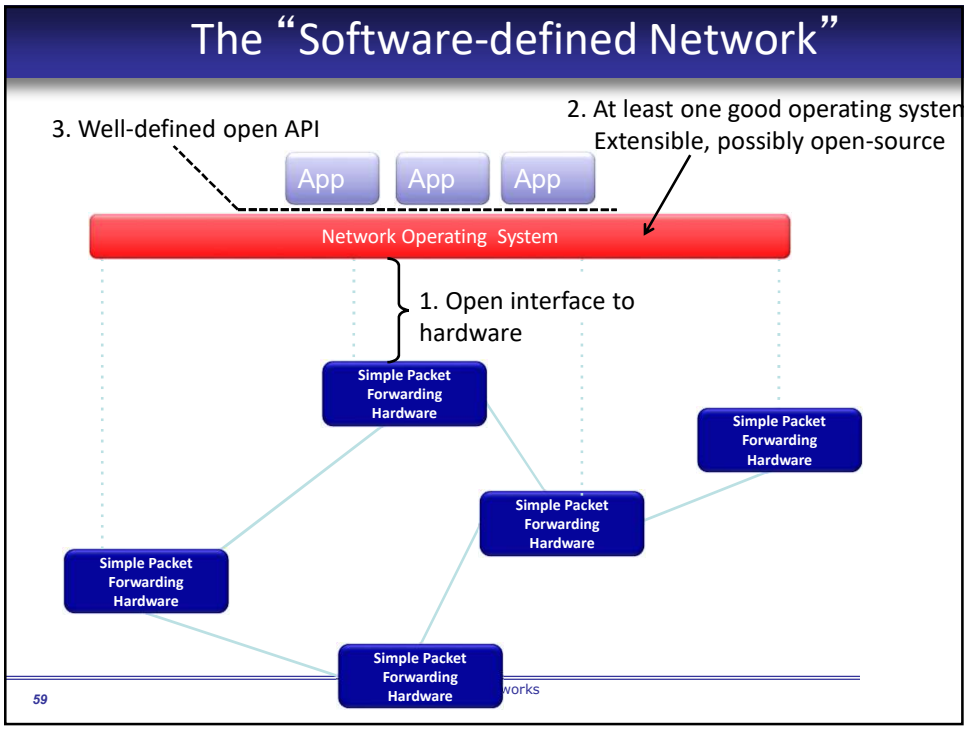
We Have Achieved Modularity!

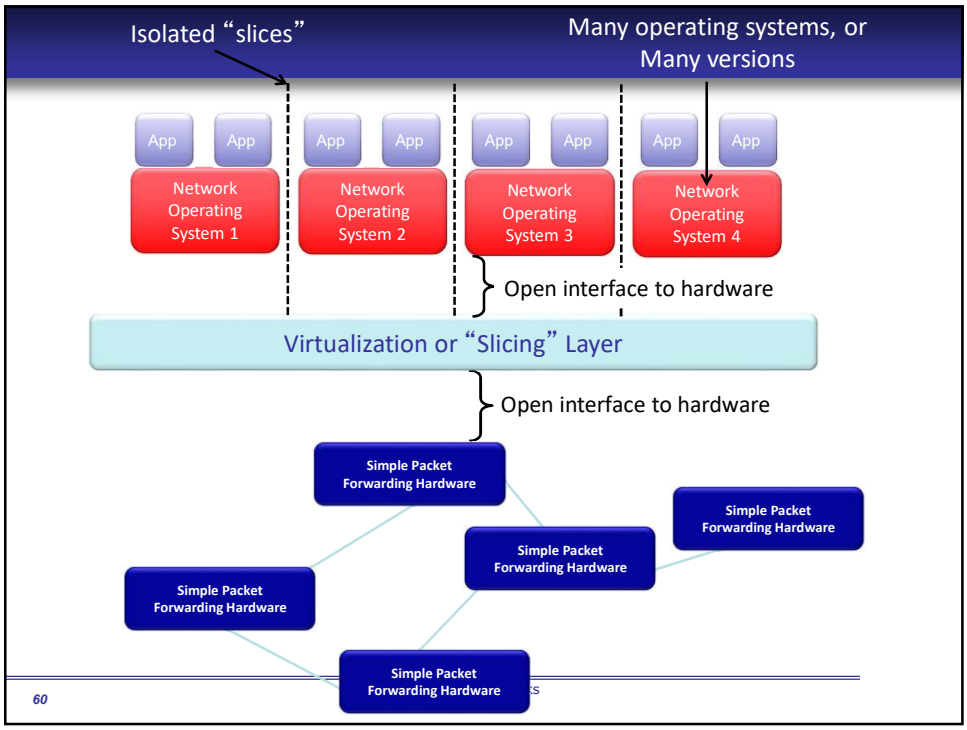
- Modularity enables independent innovation
  - Gives rise to a thriving ecosystem
- Innovation is the true value proposition of SDN
  - SDN doesn't allow you to do the impossible
  - It just allows you to do the possible much more easily
- ***This is why SDN is the future of networking…***

Is change likely?

Change is happening in non-traditional markets




The "Software-defined Network"
3. Well-defined open API
2. At least one good operating syster
Extensible, possibly open-source
App
App
App
Network Operating System
1. Open interface to hardware
Simple Packet Forwarding Hardware
Simple Packet Forwarding Hardware
Simple Packet Forwarding Hardware
Simple Packet Forwarding Hardware
Simple Packet Forwarding Hardware
59


Isolated "slices"
Many operating systems, or
Many versions
App
Network Operating System 1
Network Operating System 2
Network Operating System 3
Network Operating System 4
Open interface to hardware
Virtualization or "Slicing" Layer
Open interface to hardware
Simple Packet Forwarding Hardware
Simple Packet Forwarding Hardware
Simple Packet Forwarding Hardware
Simple Packet Forwarding Hardware
Simple Packet Forwarding Hardware
60

Consequences

More innovation in network services

- Owners, operators, 3rd party developers, researchers can improve the network
- E.g. energy management, data center management, policy routing, access control, denial of service, mobility

Lower barrier to entry for competition

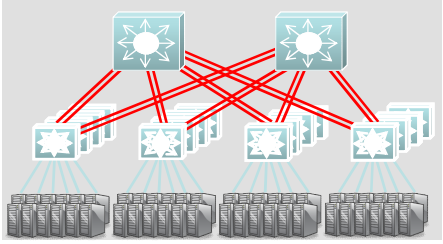
- Healthier market place, new players

The change has already started

In a nutshell

- Driven by cost and control
- Started in data centers…. and may spread
- Trend is towards an open-source, software-defined network
- Growing interest for cellular and telecom networks

Example: New Data Center



Cost

200,000 servers
Fanout of 20 ⇨ 10,000 switches
$5k commercial switch ⇨ $50M
$1k custom-built switch ⇨ $10M

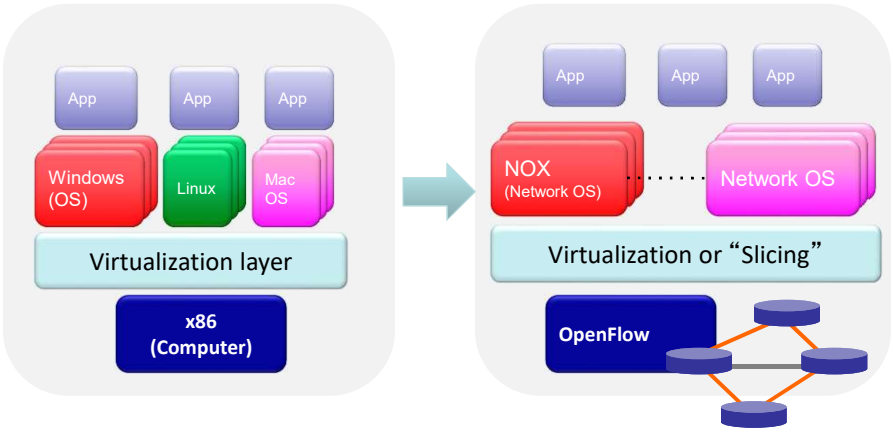
Savings in 10 data centers = $400M

Control

1. Optimize for features needed
2. Customize for services & apps
3. Quickly improve and innovate

Large data center operators are moving towards defining their own network in software.

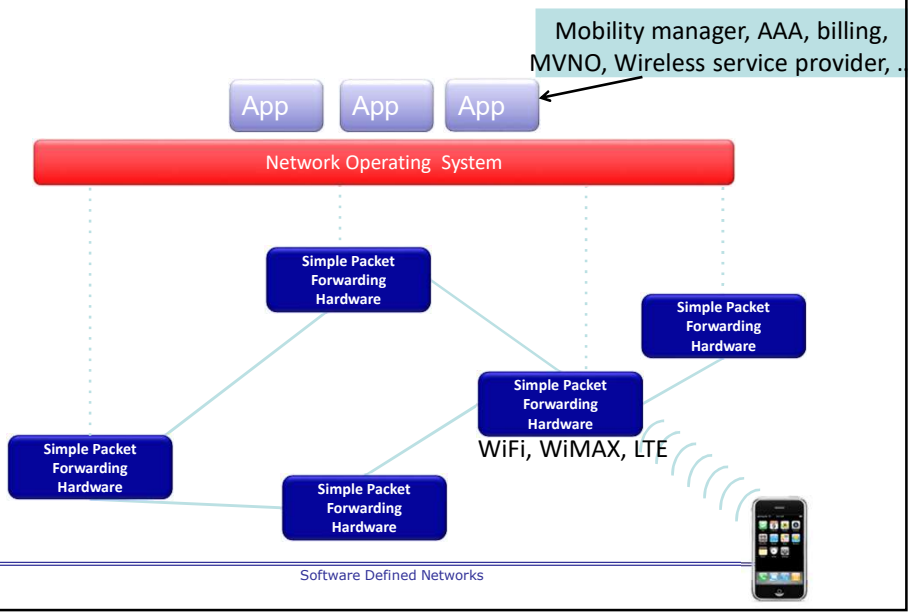
Trend



Computer Industry

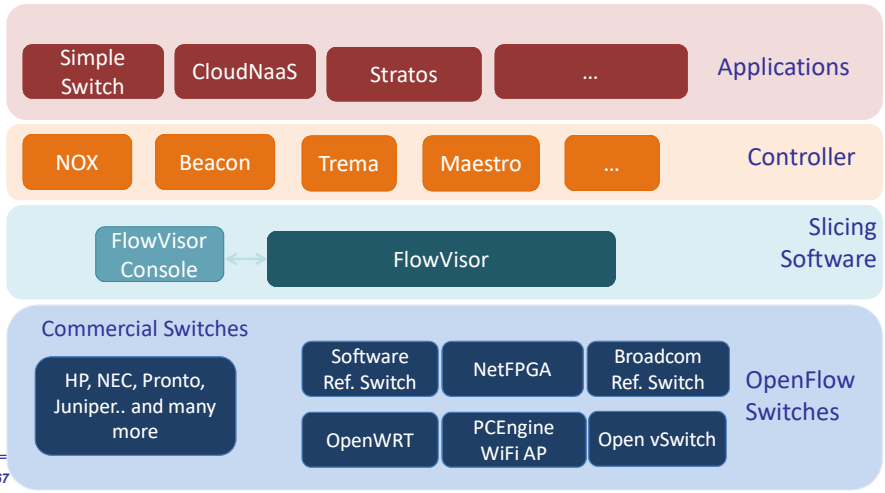
Network Industry

Software-defined Wireless Networks



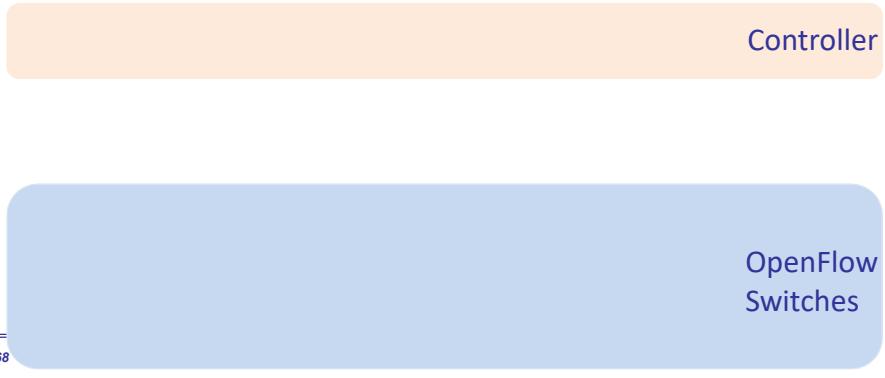
The SDN "Stack"

The SDN Stack



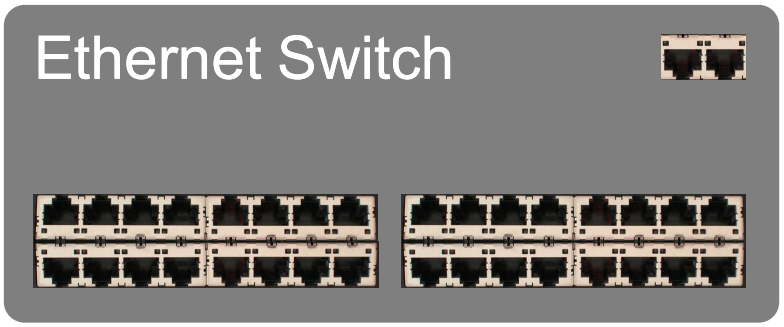
67

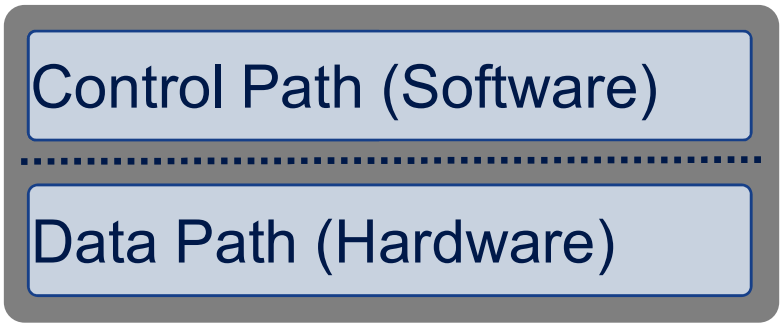
The SDN Stack

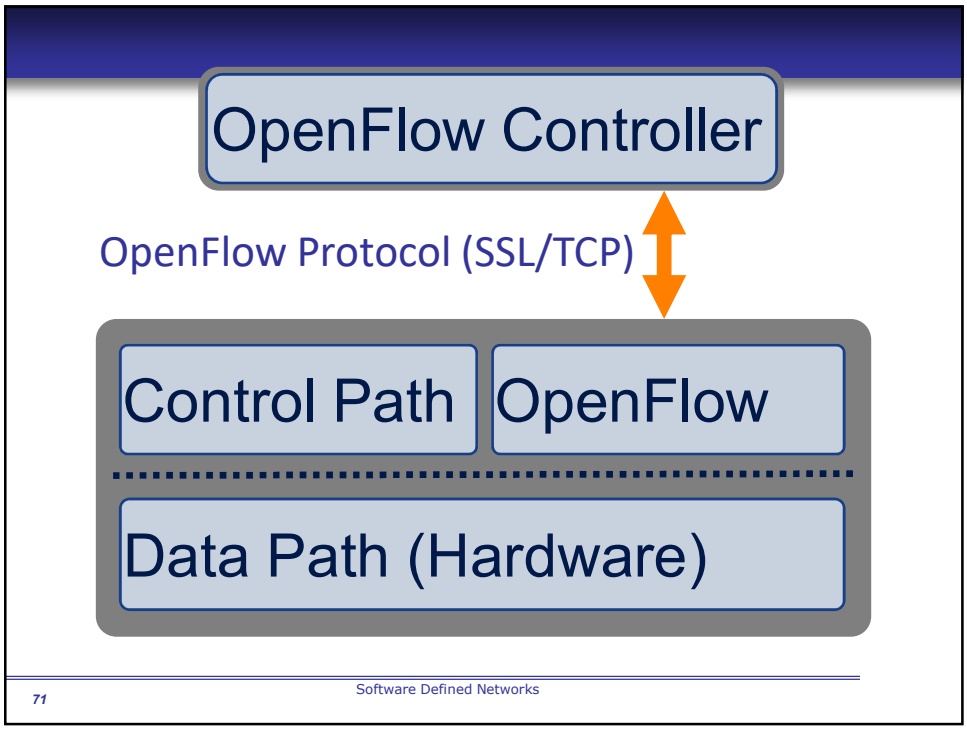


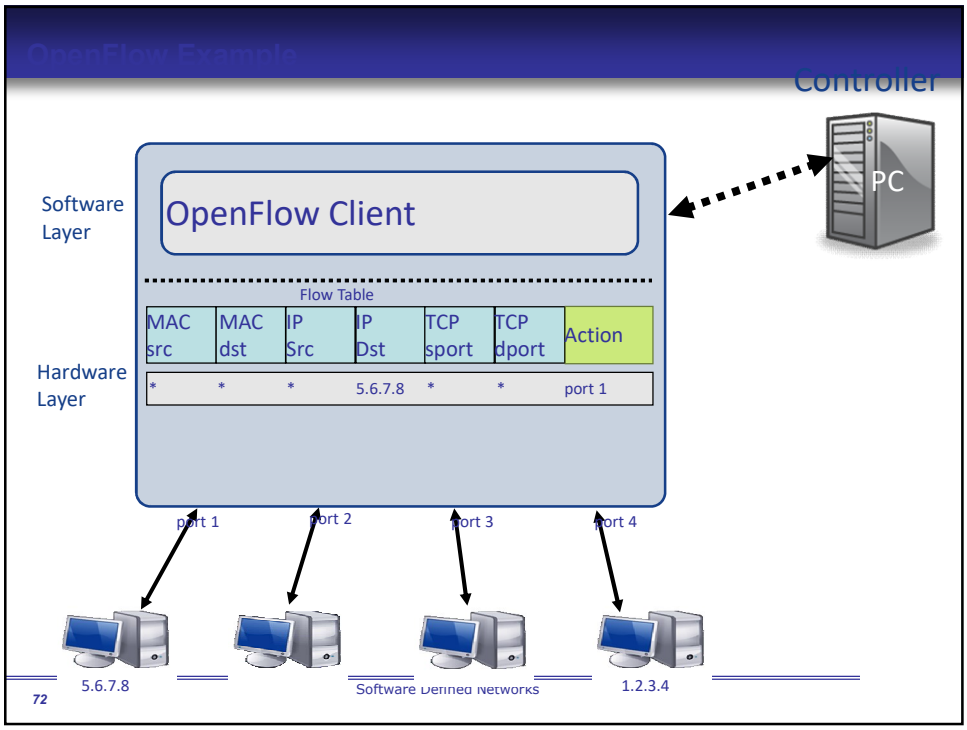
68

How does OpenFlow work?





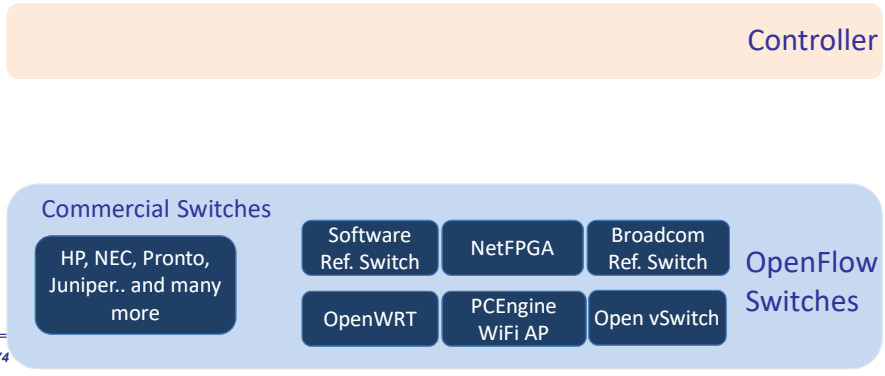

OpenFlow Controller
OpenFlow Protocol (SSL/TCP)
Control Path
OpenFlow
Data Path (Hardware)
71
Software Defined Networks


OpenFlow Example
Controller
PC
Software Layer
OpenFlow Client
Flow Table
MAC src
MAC dst
IP Src
IP Dst
TCP sport
TCP dport
Action
Hardware Layer
* * * 5.6.7.8 * * port 1
port 1
port 2
port 3
port 4
5.6.7.8
1.2.3.4
72
Software Defined Networks

OpenFlow Progression

- OF v1.0: released end of 2009: “Into the Campus”
- OF v1.1: released March 1 2011: “Into the WAN”
  - multiple tables: leverage additional tables
  - tags and tunnels: MPLS, VLAN, virtual ports
  - multipath forwarding: ECMP, groups
- OF v1.2: approved Dec 8 2011: “Extensible Protocol”
  - extensible match
  - extensible actions
  - IPv6
  - multiple controllers

The SDN Stack

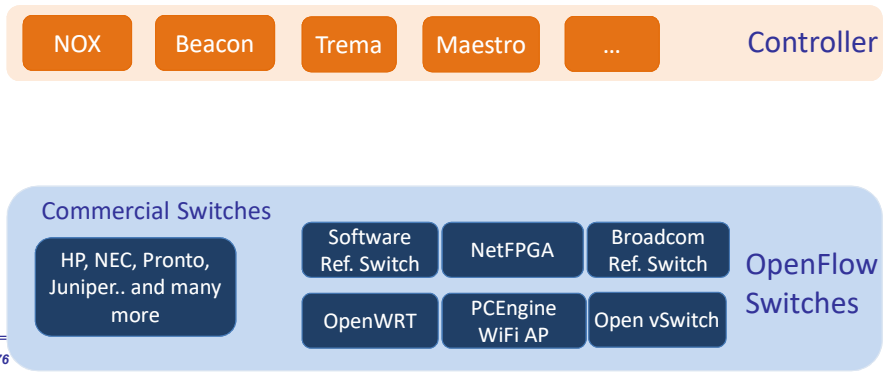


Switches

Vendor	Models	Virtualize?	Notes	Image
HP ProCurve	**5400zl, 6600, +**	1 OF instance per VLAN	-LACP, VLAN and STP processing before OF -Wildcard rules or non-IP pkts processed in s/w -Header rewriting in s/w -CPU protects mgmt during loop	
Pronto/ Pica8	3290, 3780, 3920, +	1 OF instance per switch	-No legacy protocols (like VLAN and STP) -Most actions processed in hardware -MAC header rewriting in h/w	

Name	Lang	Platform(s)	Original Author	Notes
OpenFlow Reference	C	Linux	Stanford/Nicira	not designed for extensibility
Open vSwitch	C/ Python	Linux/BSD?	Ben Pfaff/Nicira	In Linux kernel 3.3+
Indigo	C/Lua	Linux-based Hardware Switches	Dan Talayco/BigSwitch	Bare OpenFlow switch

The SDN Stack



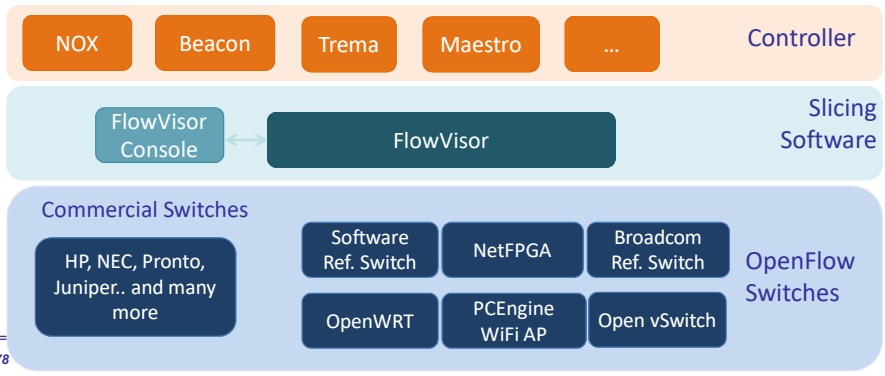
Controllers

Name	Lang	Original Author	Notes
OpenFlow Reference	C	Stanford/Nicira	not designed for extensibility
NOX	**Python, C++**	Nicira	actively developed
Beacon	Java	David Erickson (Stanford)	runtime modular, web UI framework, regression test framework
Maestro	Java	Zheng Cai (Rice)	
Trema	Ruby, C	NEC	includes emulator, regression test framework
RouteFlow	?	CPqD (Brazil)	virtual IP routing as a service
POX	**Python**		
Floodlight	Java	BigSwitch, based on Beacon	

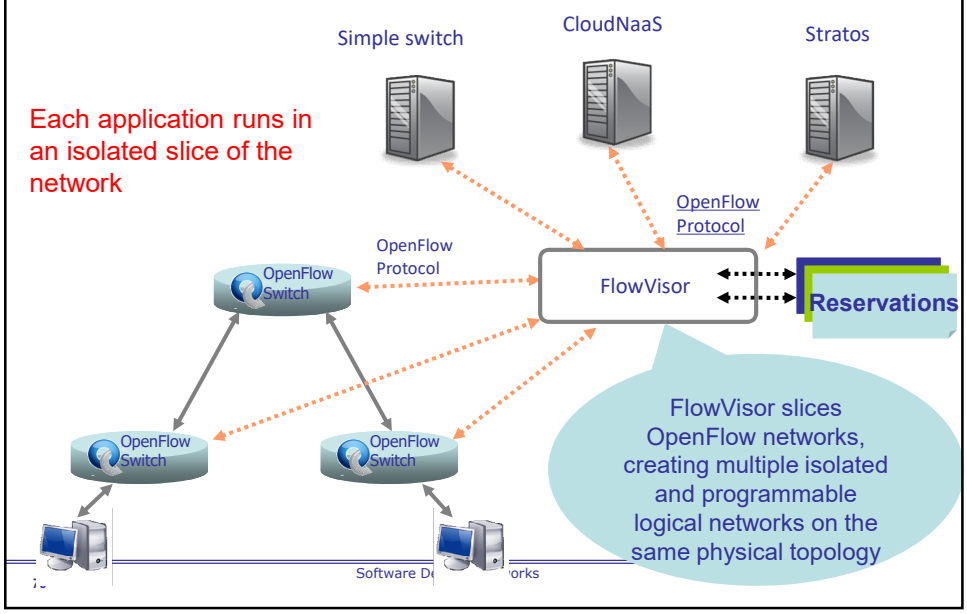
Too many to easily keep track of…

http://yuba.stanford.edu/~casado/of-sw.html

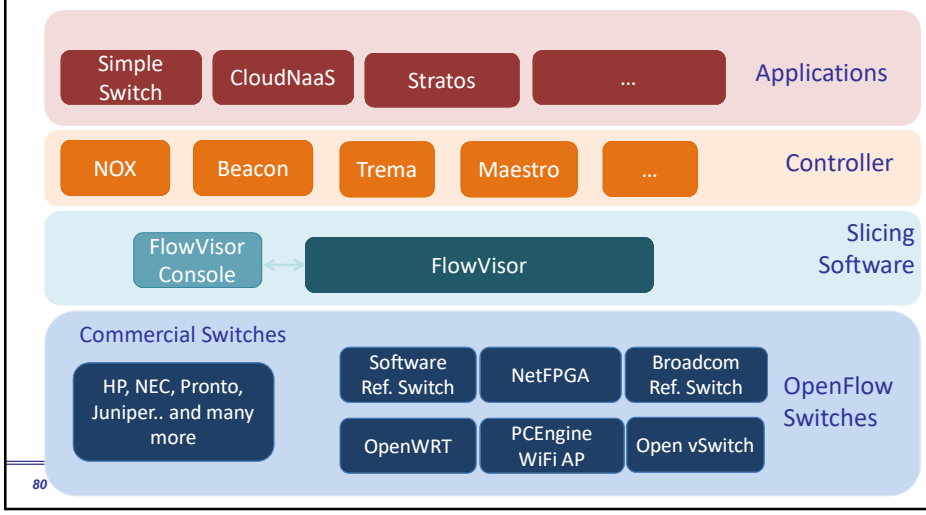
The SDN Stack



FlowVisor Creates Virtual Networks



The SDN Stack



Example SDN Applications

Wisconsin Projects

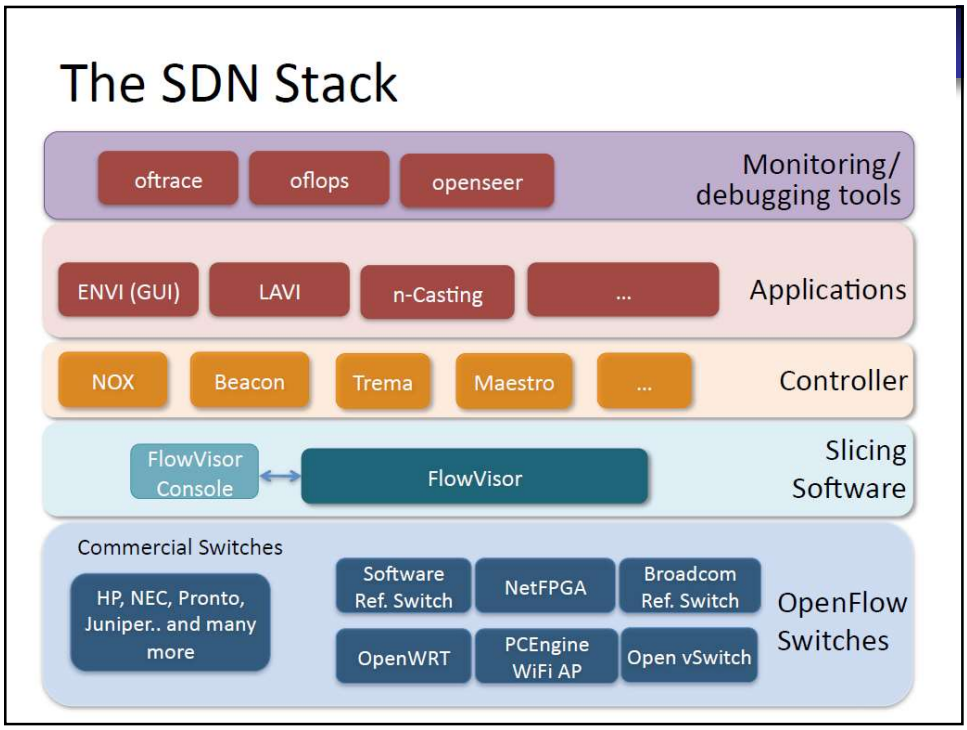
- Stratos
- CloudNaaS
- OpenSAFE
- ECOS

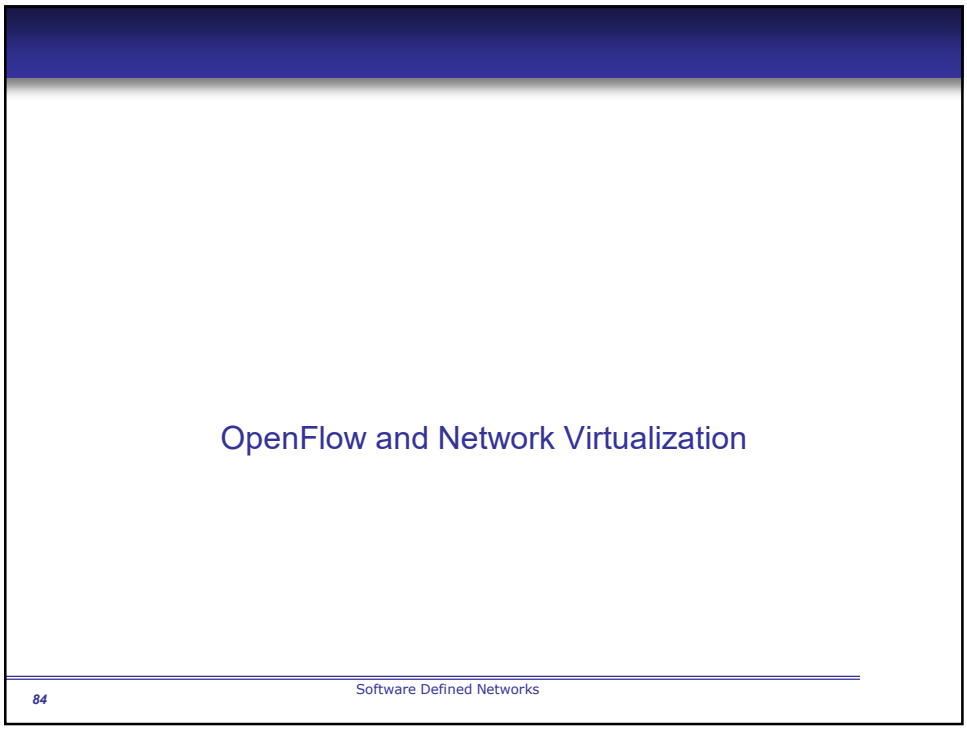
Stanford Demos

- Wireless mobility
- VM mobility/migration
- Network virtualization
- Power management
- Load balancing
- Traffic Engineering

openflow.org/videos

82


The SDN Stack
oftrace
oflops
openseer
Monitoring/ debugging tools
ENVI (GUI)
LAVI
n-Casting
...
Applications
NOX
Beacon
Trema
Maestro
...
Controller
FlowVisor Console
FlowVisor
Slicing Software
Commercial Switches
HP, NEC, Pronto, Juniper.. and many more
Software Ref. Switch
NetFPGA
Broadcom Ref. Switch
OpenWRT
PCEngine WiFi AP
Open vSwitch
OpenFlow Switches


OpenFlow and Network Virtualization
84
Software Defined Networks

In a nutshell

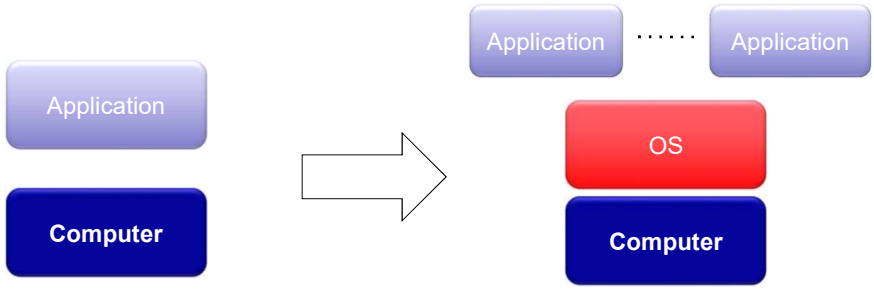
- A revolution is just starting in networking
  - Driven by cost and control
  - It started in data centers…. and is spreading
  - Trend is towards an open-source, software-defined network
- The new opportunity to innovate will bring about the need to try new ideas
  - Hence virtualization (or slicing)
- Outline one way to do it with OpenFlow

Software-defined Network

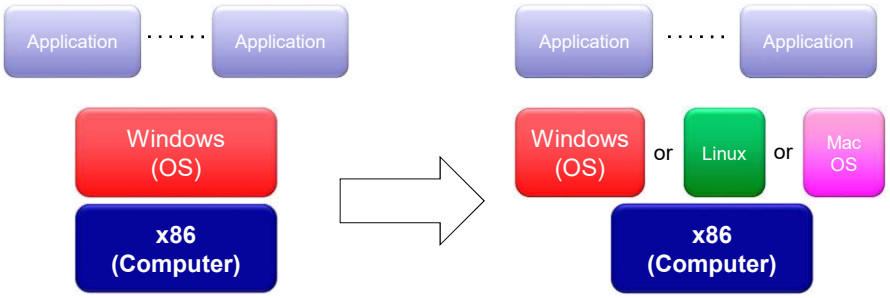
1. Data Centers
   - Cost and control
2. Network & Cellular operators
   - Bit-pipe avoidance
   - Cost and control
   - Security and mobility

1. Researchers
   - GENI, FIRE, …

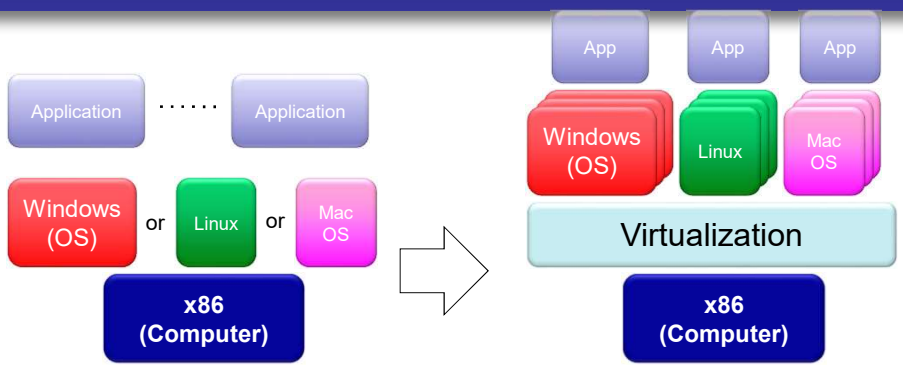
What form might it take?



OS abstracts hardware substrate
→ Innovation in applications



Simple, common, stable, hardware substrate below
+ Programmability
+ Competition
→ Innovation in OS and applications



Simple, common, stable, hardware substrate below
+ Programmability
+ Strong isolation model
+ Competition above
→ Innovation in infrastructure

A simple stable common substrate

1. Allows applications to flourish
   Internet: Stable IPv4 led to the web
2. Allows the infrastructure on top to be defined in software
   Internet: Routing protocols, management, …
3. Rapid innovation of the infrastructure itself
   Internet: er...? What's missing? What is the substrate…?

(Statement of the obvious)

- In networking, despite several attempts…
- Never agreed upon a clean separation between:
  1. A simple common hardware substrate
  2. And an open programming environment on top

A prediction

1. A clean separation between the substrate and an open programming environment
2. A simple low-cost hardware substrate that generalizes, subsumes and simplifies the current substrate
3. Very few preconceived ideas about how the substrate will be programmed
4. Strong isolation among features

- But most of all….

Open-source will play a large role

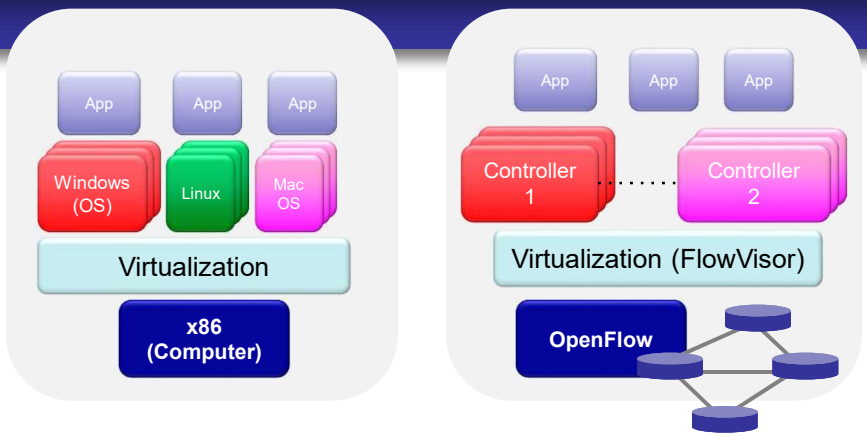
Owners, operators, administrators, developers, researchers will want to…

…improve, update, fix, experiment, share,
build-upon, and version
their network.

Therefore, the software-defined network will allow simple ways to program and version.

One way to do this is virtualizing/slicing the network substrate.

OpenFlow as a simple, sliceable substrate below

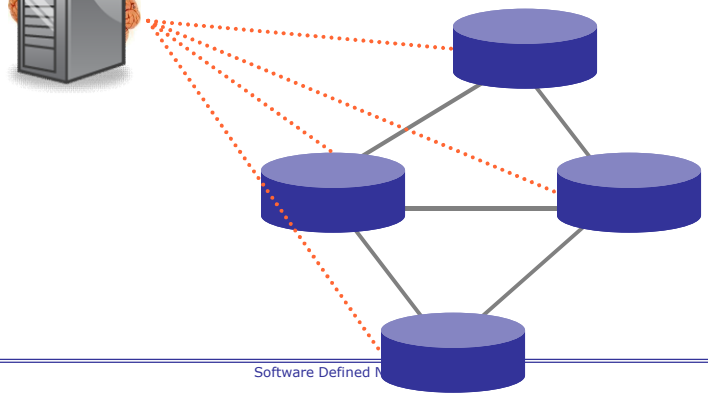


Simple, common, stable, hardware substrate below
+ Programmability
+ Strong isolation model
+ Competition above
→ Faster innovation

Step 1:
Separate intelligence from datapath

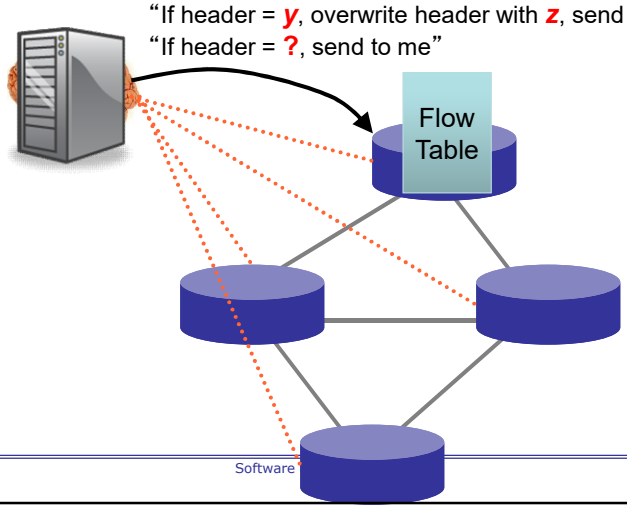
Operators, users, 3rd party developers, researchers, …

New function!

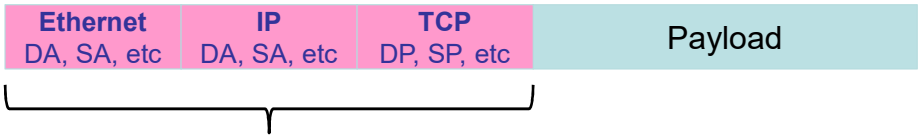


Step 2: Cache decisions in minimal flow based datapath

"If header = ***x***, send to port 4"
"If header = ***y***, overwrite header with ***z***, send to ports 5,6"
"If header = **?**, send to me"



Packet-switching substrate

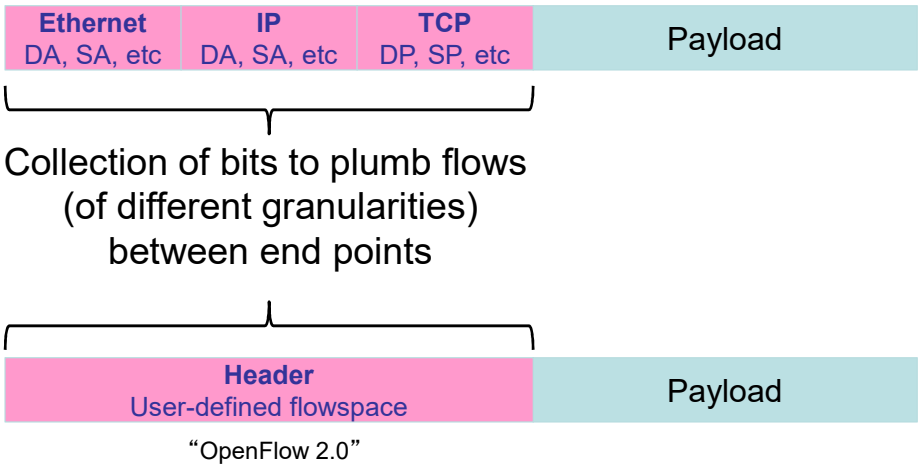


Collection of bits to plumb flows (of different granularities) between end points

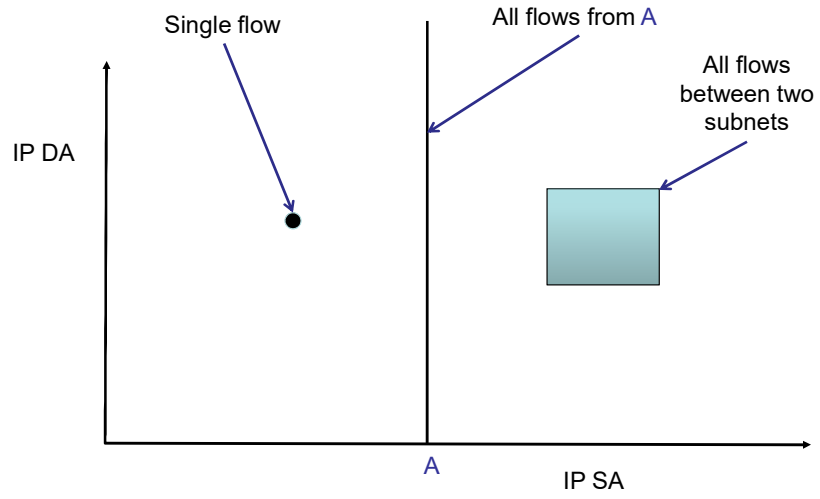
Properties of a flow-based substrate

- 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)

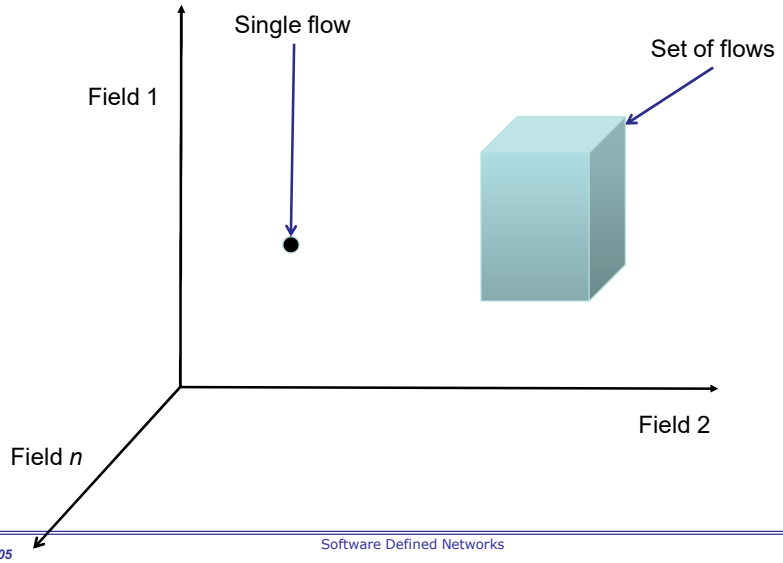
Substrate: "Flowspace"



Flowspace: Simple example



Flowspace: Generalization



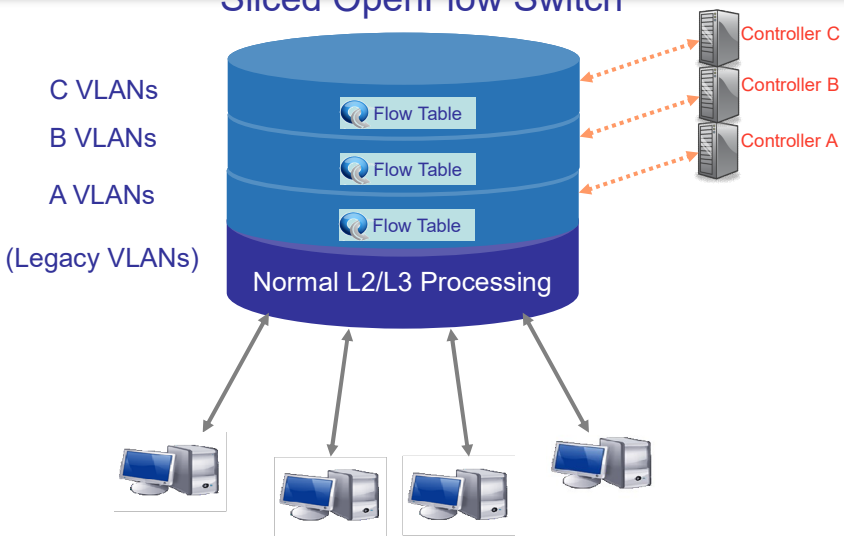
Properties of Flowspace

- 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

Slicing Flowspace

Approach 1: Slicing using VLANs

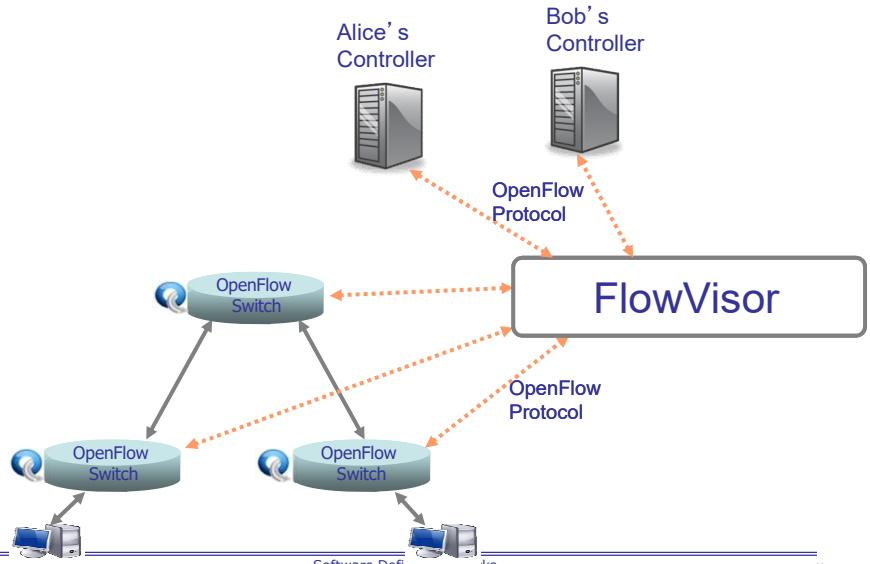
Sliced OpenFlow Switch



Some prototype OpenFlow switches do this…

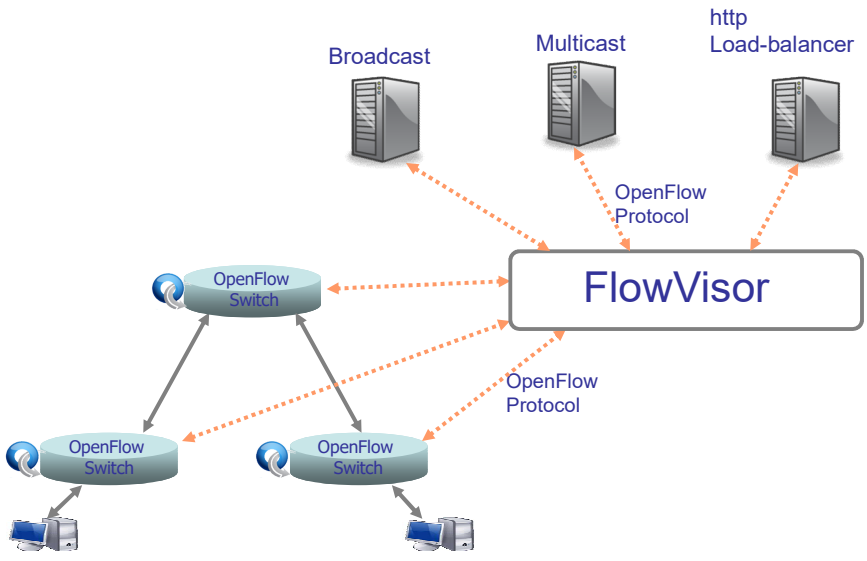
Approach 2: FlowVisor

Rob Sherwood* (rob.sherwood@stanford.edu)

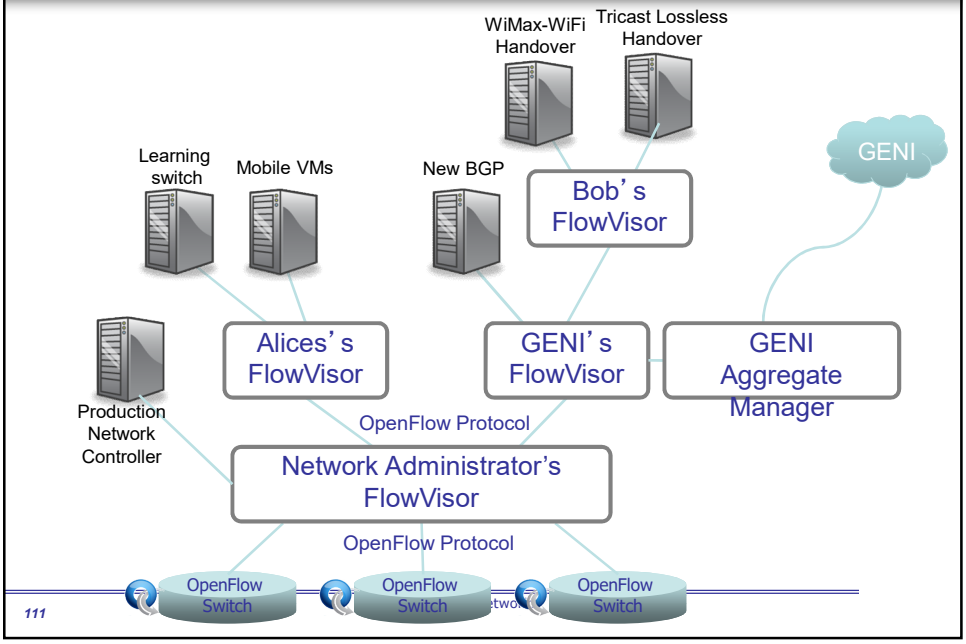


* Deutsche Telekom, "T-Labs"

FlowVisor



FlowVisor



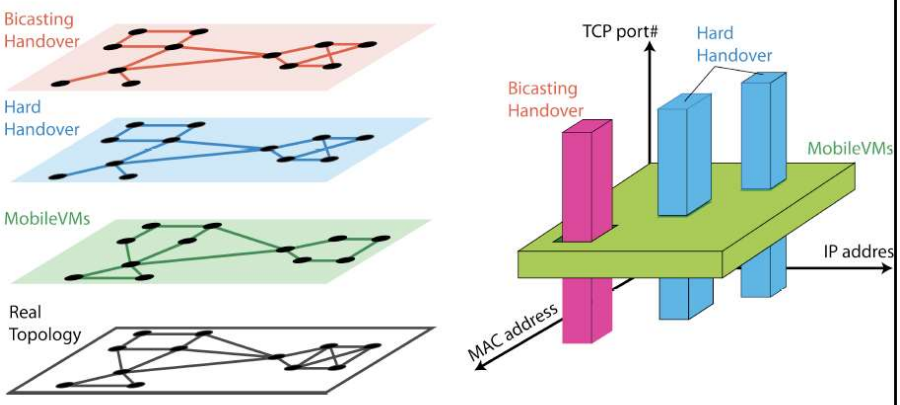
FlowVisor

- A proxy between switch and guest controller
- Parses and rewrites OpenFlow messages as they pass
- Ensures that one experiment does not affect another
- Allows rich virtual network boundaries
  - By port, by IP, by flow, by time, etc.
- Define virtualization rules in software

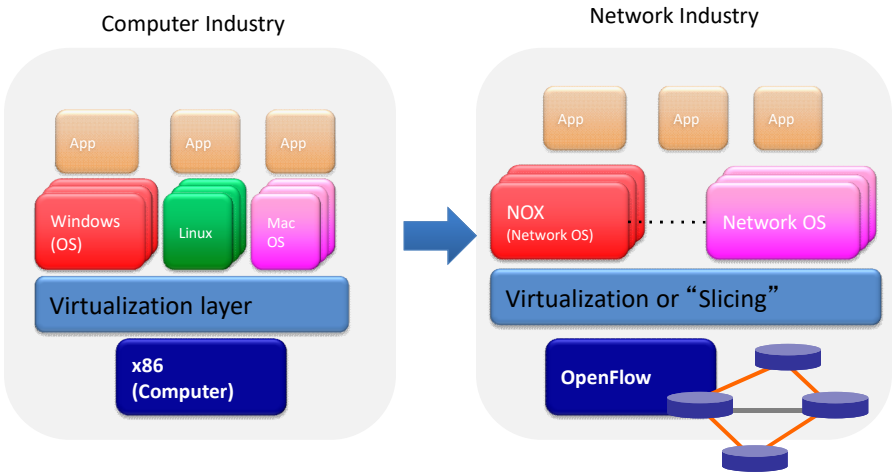
FlowVisor Goals

- Transparency
  - Unmodified guest controllers
  - Unmodified switches
- Strong resource Isolation
  - Link b/w, switch CPU, etc.
  - Flow space: who gets this message
- Virtualization Policy module
- Rich network slicing

Slicing Example



Trend

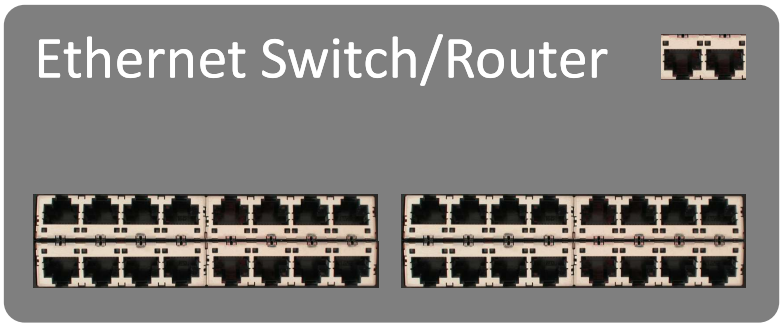


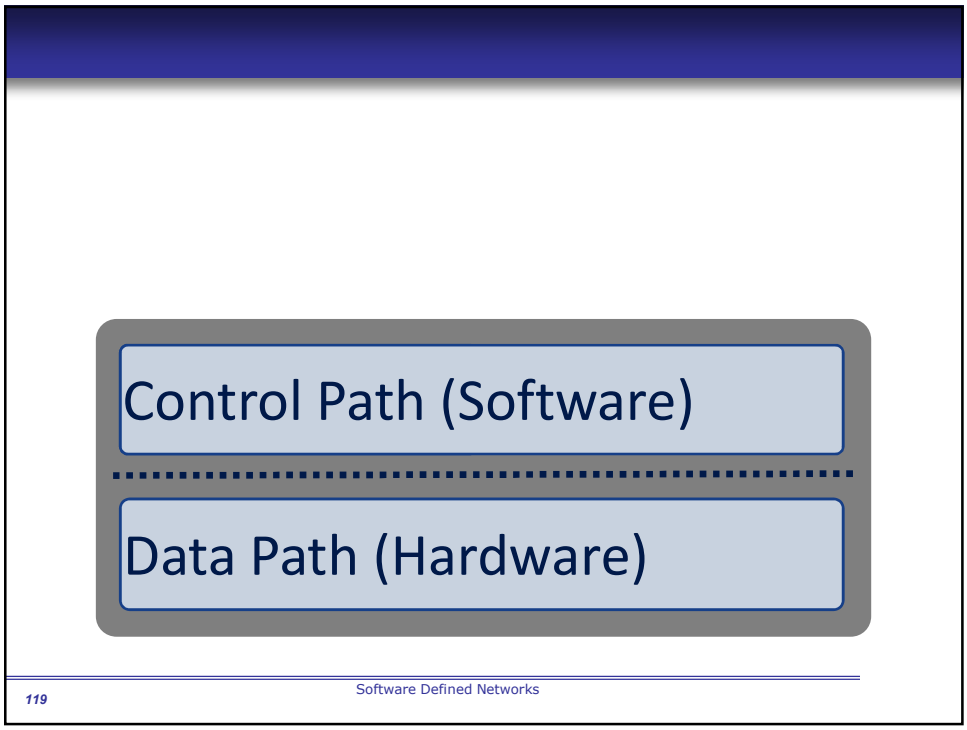
Simple common stable hardware substrate below+ programmability + strong isolation model + competition above = Result : faster innovation

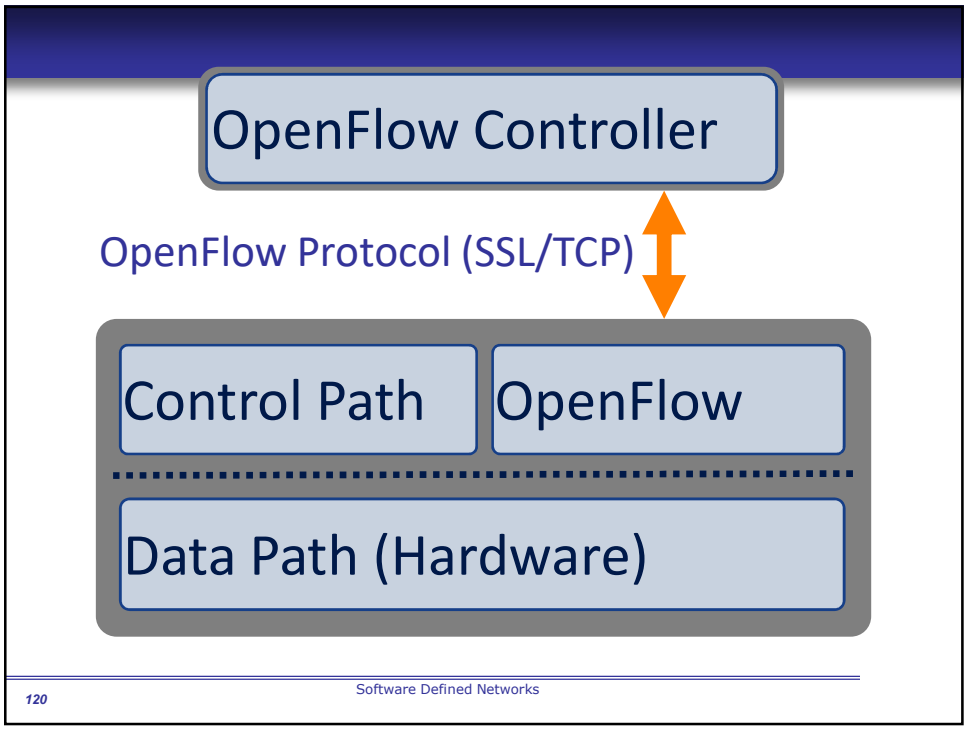
What is OpenFlow?

Short Story: OpenFlow is an API

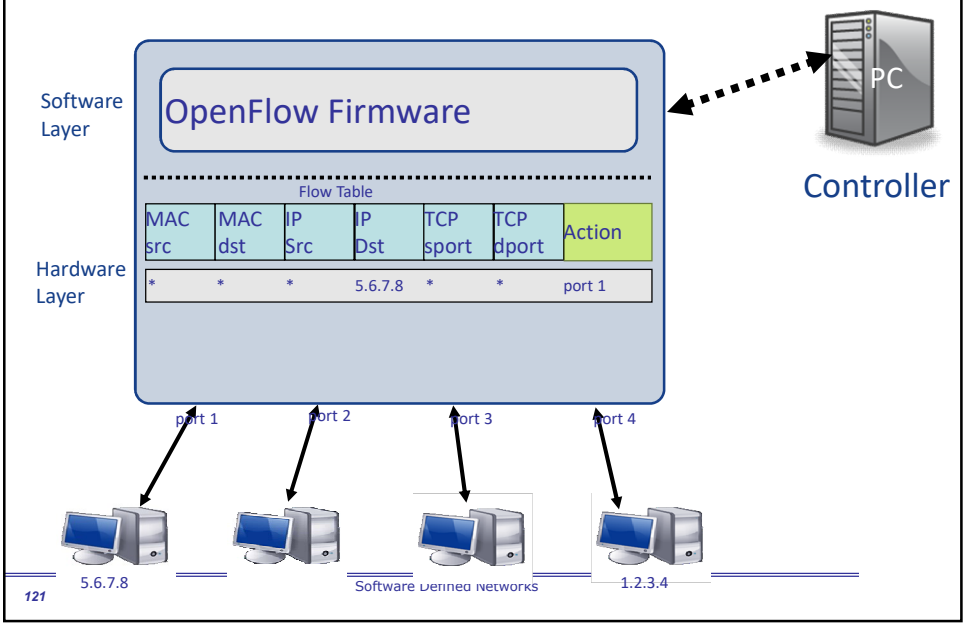
- Control how packets are forwarded
- Implementable on COTS hardware
- Make deployed networks programmable
  - not just configurable
- Makes innovation easier
- Result:
  - Increased control: custom forwarding
  - Reduced cost: API → increased competition



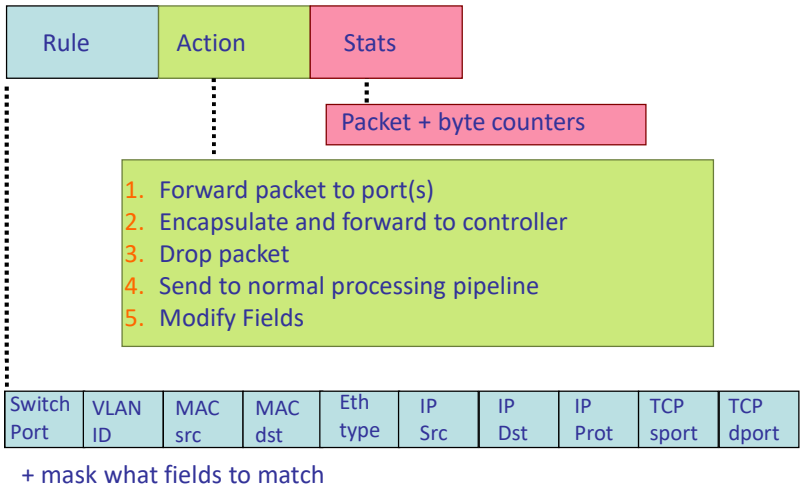

Control Path (Software)
Data Path (Hardware)


OpenFlow Controller
OpenFlow Protocol (SSL/TCP)
Control Path
OpenFlow
Data Path (Hardware)

OpenFlow Flow Table Abstraction



OpenFlow Basics - Flow Table Entries



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	Forward
*	*	*	*	*	*	*	*	*	22	drop

Examples

Routing

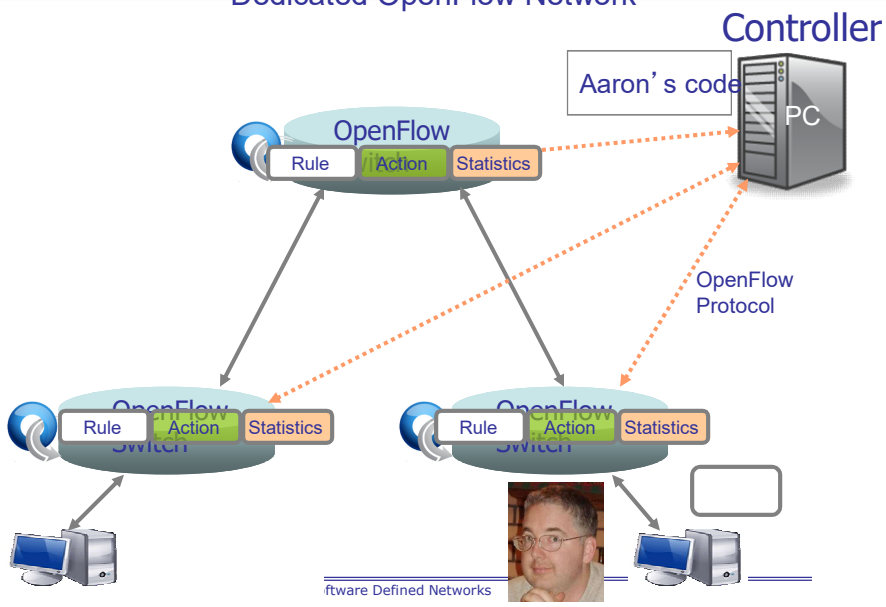
Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	*	*	*	*	5.6.7.8	*	*	*	port6

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 Usage

Dedicated OpenFlow Network



Network Design Decisions

Forwarding logic (of course)

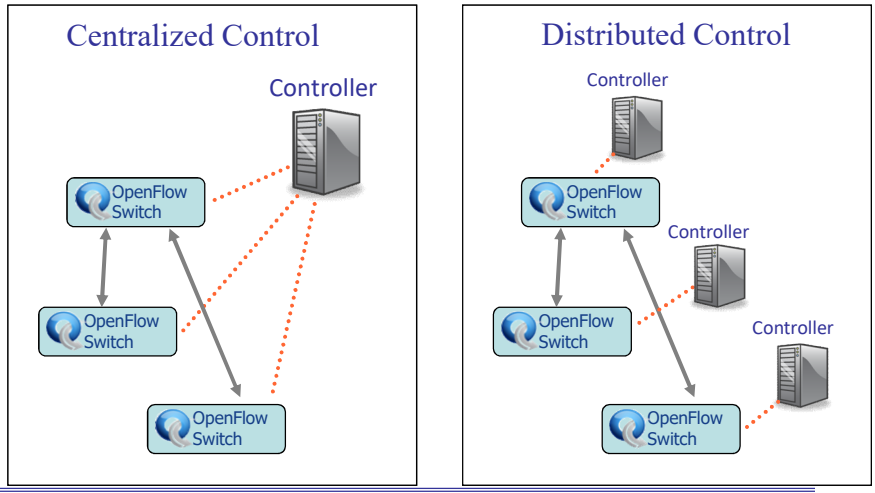
Centralized vs. distributed control

Fine vs. coarse grained rules

Reactive vs. Proactive rule creation

Likely more: open research area

Centralized vs Distributed Control



Flow Routing vs. Aggregation

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

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

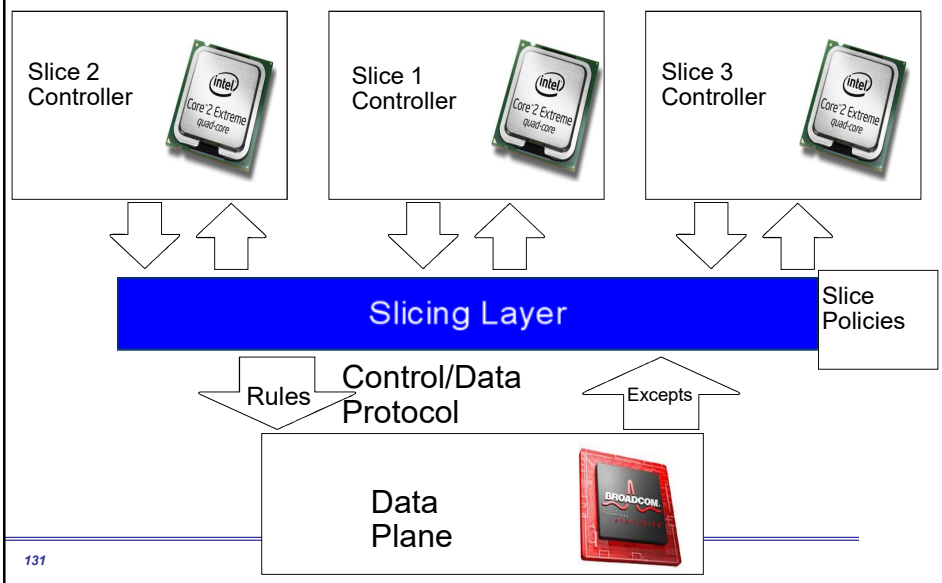
Proactive

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 Application: Network Slicing

- Divide the production network into logical *slices*
  - each slice/service controls its own packet forwarding
  - users pick which slice controls their traffic: opt-in
  - existing production services run in their own slice
    - e.g., Spanning tree, OSPF/BGP
- Enforce strong isolation between slices
  - actions in one slice do not affect another
- Allows the (logical) testbed to mirror the production network
  - real hardware, performance, topologies, scale, users
- Prototype implementation: FlowVisor

Add a Slicing Layer Between Planes



Network Slicing Architecture

➢ A network slice is a collection of sliced switches/routers

- Data plane is unmodified
  - Packets forwarded with no performance penalty
  - Slicing with existing ASIC
- Transparent slicing layer
  - each slice believes it owns the data path
  - enforces isolation between slices
    - i.e., rewrites, drops rules to adhere to slice police
  - forwards exceptions to correct slice(s)

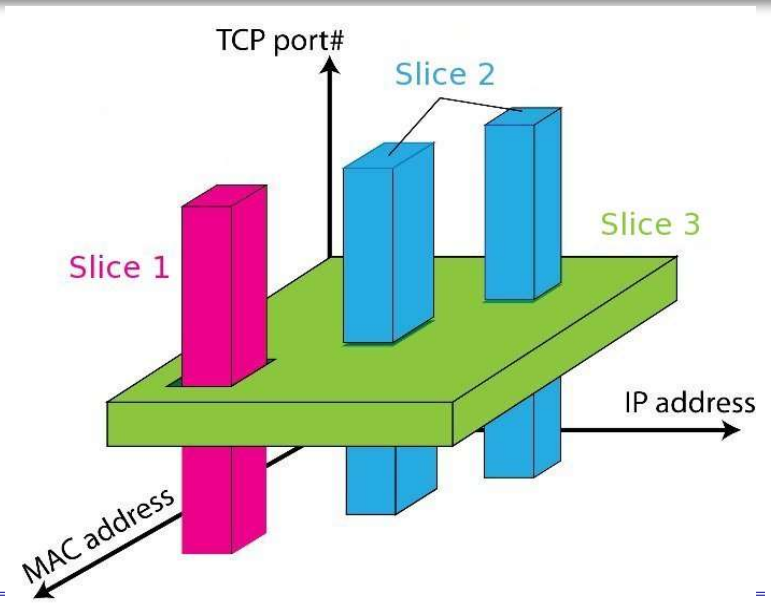
Slicing Policies

- The policy specifies resource limits for each slice:
  - Link bandwidth
  - Maximum number of forwarding rules
  - Topology
  - Fraction of switch/router CPU
  - *FlowSpace: which packets does the slice control?*

Flowspace

- Flowspace is a way of thinking about classes of packets
- Each slice has forwarding control of a specific set of packets, as specified by packet header fields
  - all packets in a given flow are controlled by the same slice
- Each flow is controlled by exactly one slice (ignoring monitoring slices)
- In practice, flow spaces are described using ordered ACL-like rules

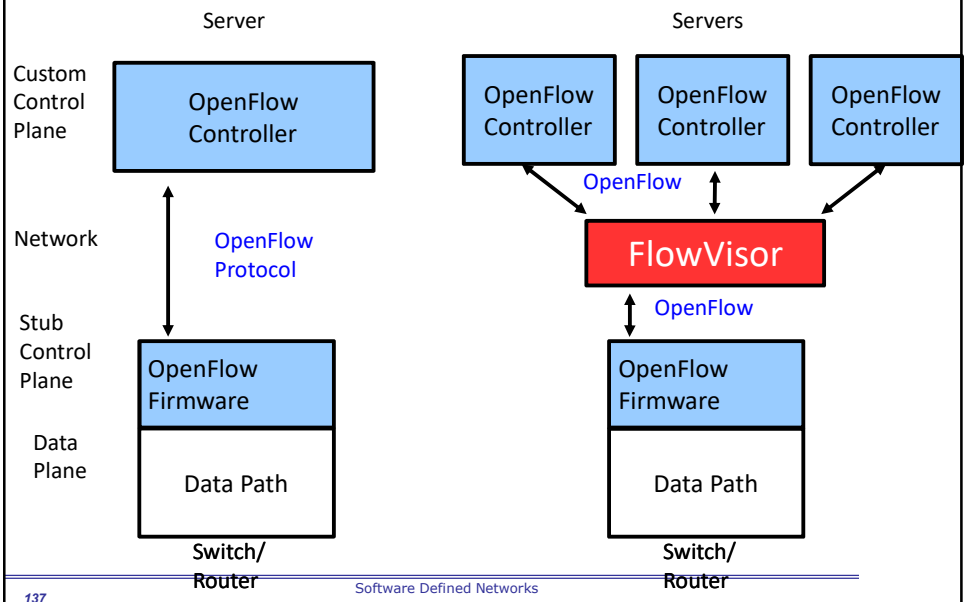
FlowSpace: Maps Packets to Slices



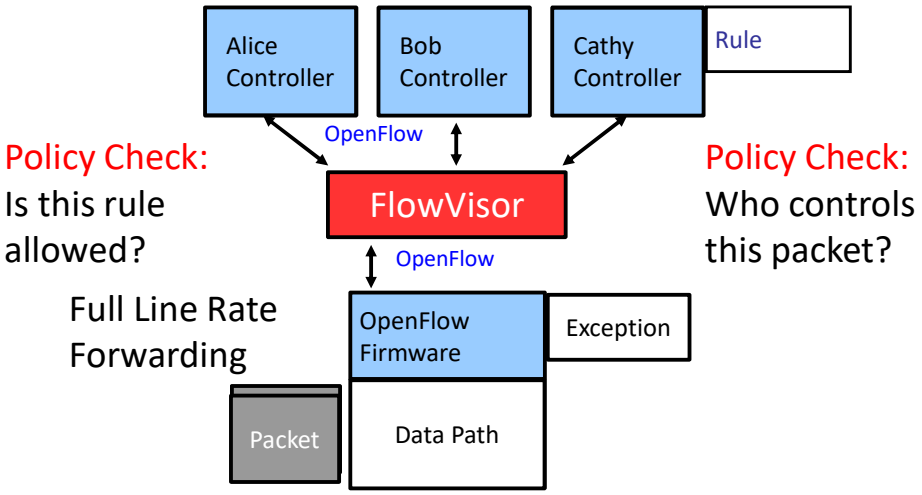
Real User Traffic: Opt-In

- Allow users to Opt-In to services in real-time
  - Users can delegate control of individual flows to Slices
  - Add new FlowSpace to each slice's policy
- Example:
  - *"Slice 1 will handle my HTTP traffic"*
  - *"Slice 2 will handle my VoIP traffic"*
  - *"Slice 3 will handle everything else"*
- Creates incentives for building high-quality services

FlowVisor Implemented on OpenFlow



FlowVisor Message Handling



OpenFlow Deployments

OpenFlow has been prototyped on….

- Ethernet switches
  - HP, Cisco, NEC, Quanta, + more underway
- IP routers
  - Cisco, Juniper, NEC
- Switching chips
  - Broadcom, Marvell
- Transport switches
  - Ciena, Fujitsu
- WiFi APs and WiMAX Basestations

Most (all?) hardware switches now based on Open vSwitch…

Deployment: Stanford

- Real, production network
  - 15 switches, 35 APs
  - 25+ users
  - 1+ year of use
- Same physical network hosts Stanford demos
  - 7 different demos

Deployments: GENI

(Public) Industry Interest

- Google has been a main proponent of new OpenFlow 1.1 WAN features
  - ECMP, MPLS-label matching
  - MPLS LDP-OpenFlow speaking router: NANOG50
- NEC has announced commercial products
  - Initially for datacenters, talking to providers
- Ericsson
  - "MPLS Openflow and the Split Router Architecture: A Research Approach" at MPLS2010

Conclusions

- Current networks are complicated
- OpenFlow is an API
  - Interesting apps include network slicing
- Nation-wide academic trials underway
- OpenFlow has potential for Service Providers
  - Custom control for Traffic Engineering

NOX: A Bit of History

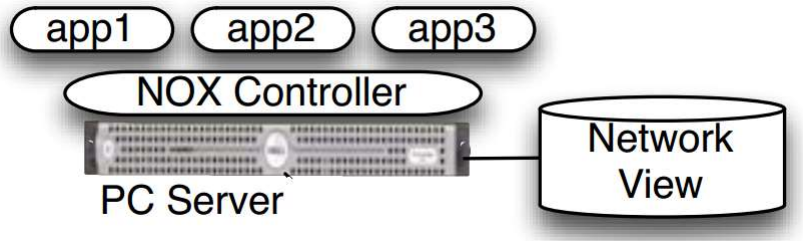
- NOX was the first SDN controller
- Released under GPL in 2008
  - Extensively used in research
- Now maintained by research community

NOX Highlights

- Linux
- C++ and Python
- Component system
- Event-based programming model
- Applications:
  - Forwarding (reactive), topology discovery, host tracking, …

NOX

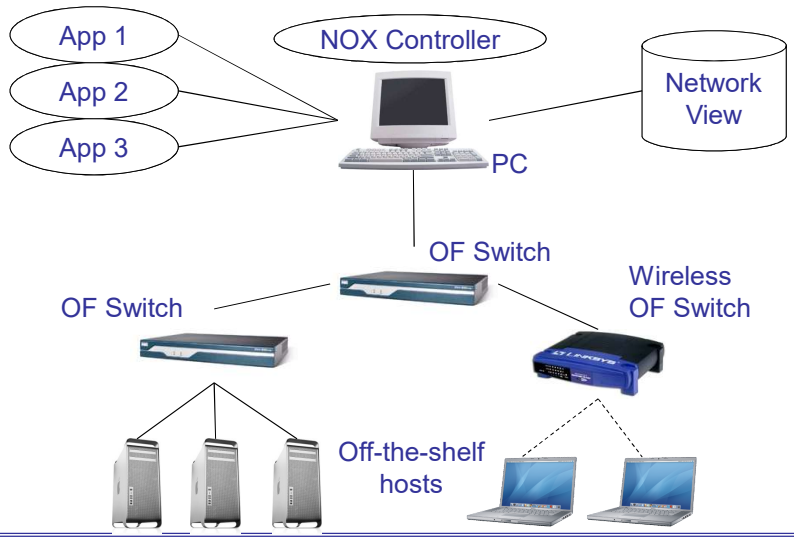
- Centralized programming model
- High-level abstraction



Programming Interface

- Events
- Namespace
- Libraries
  - Routing
  - Packet classification
  - DNS
  - Network filtering

NOX Architecture



Switch Abstraction

OpenFlow switch abstraction is a flow table.

Each flow table entry takes the form

<header : counters, actions>

Switch executes the actions corresponding to the highest-priority matching header in table.

Operation

Switch

1. Packet p reaches switch.
2. **If** p matches a flow entry
   Then apply the corresponding actions
   Else forward to the controller

Controller

1. Packet p reaches controller.
2. Update view of network state.
3. Decide the route for the packet and inform the relevant switches of that route.

Application I/O

Observation granularity:

- Switch-level topology
- Locations of users, hosts, middleboxes
- Services offered, e.g. HTTP or NFS
- Bindings between names and addresses
- NOT the entire packet/flow state

Control granularity: flows.

Decisions about one packet are applied to all subsequent packets in the flow.

Programmatic Interface: Events

NOX exposes network events to applications

- Switch join
- Switch leave
- User authenticated
- Flow initiated
- …

Applications consist of code fragments that respond to these events.

Example: Access Control

```
function handle_flow_initialize(packet)
    usersrc = nox.resolve_user_src(packet)
    hostsrc = nox.resolve_host_src(packet)
    usertgt = nox.resolve_user_tgt(packet)
    hosttgt = nox.resolve_host_tgt(packet)
    prot = nox.resolve_ap_prot(packet)
    if deny(usersrc,hostsrc,usertgt,hosttgt,prot) then
          nox.drop(packet)
    else nox.installpath(p, nox.computepath(p))

function deny(usersrc, hostsrc, usertgt, hosttgt, prot)
    …
```

Scalability

Events (per second)

- Packet arrivals (10^6): handled by switches
- Flow initiations (10^5) : handled by controller
- View change (10): handled by controller

Controller

- Can be replicated.
- Only global data structure: view.
- One currently handles 10^5 flow initiations per second.

Related Work

4D project (2005): provide global view of network via centralized controller.

SANE/Ethane (2007): extends 4D by adding users/nodes to the namespace and captures flow-initiation.

NOX (2008): extends SANE/Ethane

- Scaling for large networks.
- General programmatic control of network.

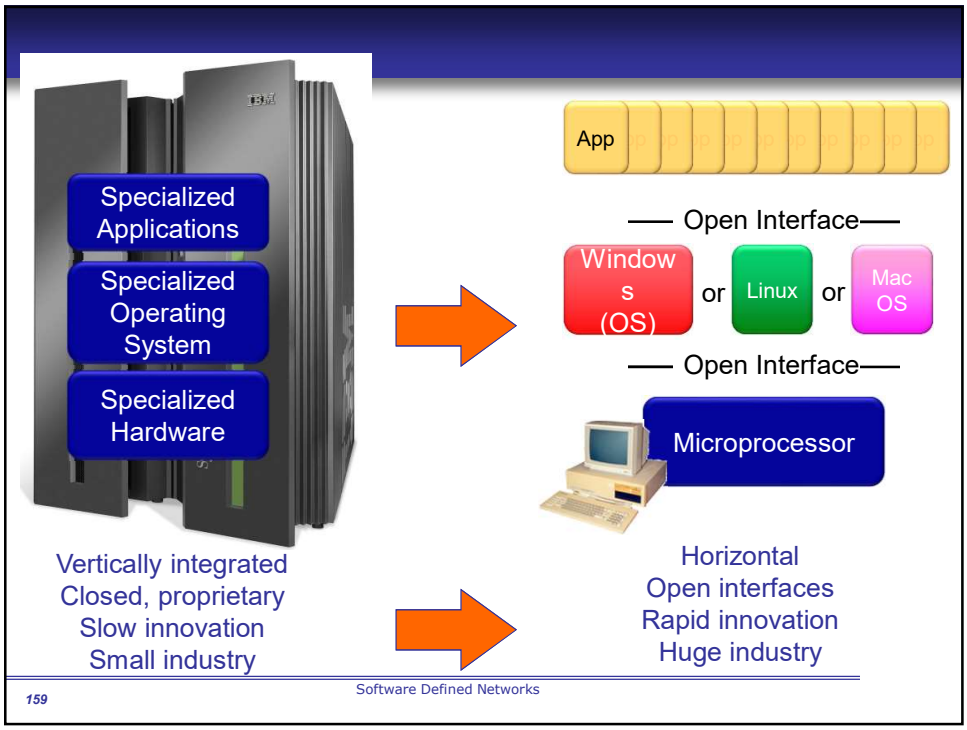
Maestro (2008): "network OS" focused on controlling interactions between applications.

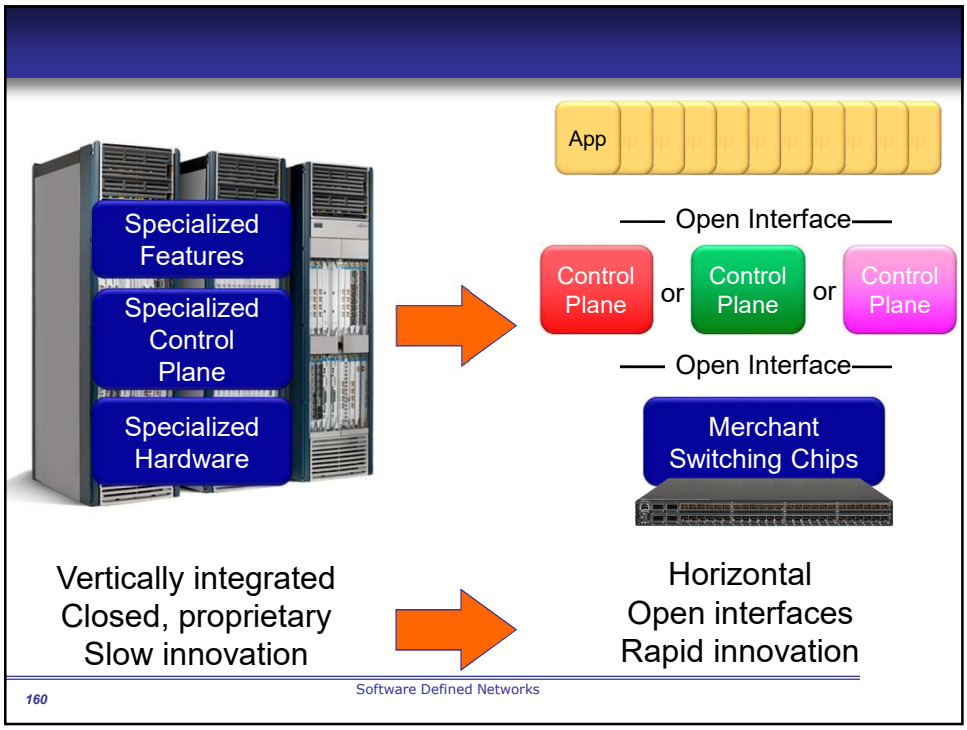
Industry: deep-packet inspection, firewalls, etc. are appliances--can be leveraged by NOX. Also, functionality similar to Ethane.

POX

- A new platform in pure Python
  - Clean dependencies
  - Take good things from NOX
  - Target Linux, Mac OS, and Windows
- Goal: Good for research
- Non-goal: Performance

Network Debugging


Specialized Applications
Specialized Operating System
Specialized Hardware
App
Open Interface
Windows (OS)
or
Linux
or
Mac OS
Open Interface
Microprocessor
Vertically integrated
Closed, proprietary
Slow innovation
Small industry
Horizontal
Open interfaces
Rapid innovation
Huge industry
159
Software Defined Networks

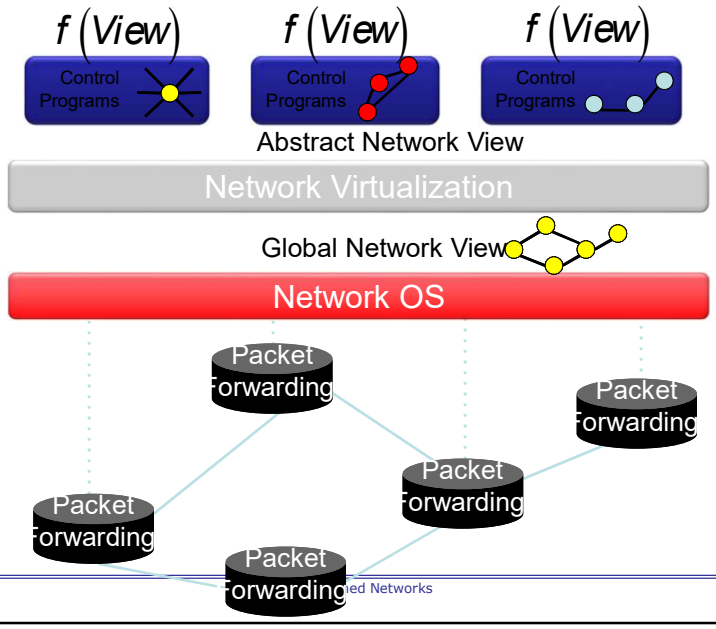

Specialized Features
Specialized Control Plane
Specialized Hardware
App
Open Interface
Control Plane
or
Control Plane
or
Control Plane
Open Interface
Merchant Switching Chips
Vertically integrated
Closed, proprietary
Slow innovation
Horizontal
Open interfaces
Rapid innovation
160
Software Defined Networks

New Research Areas

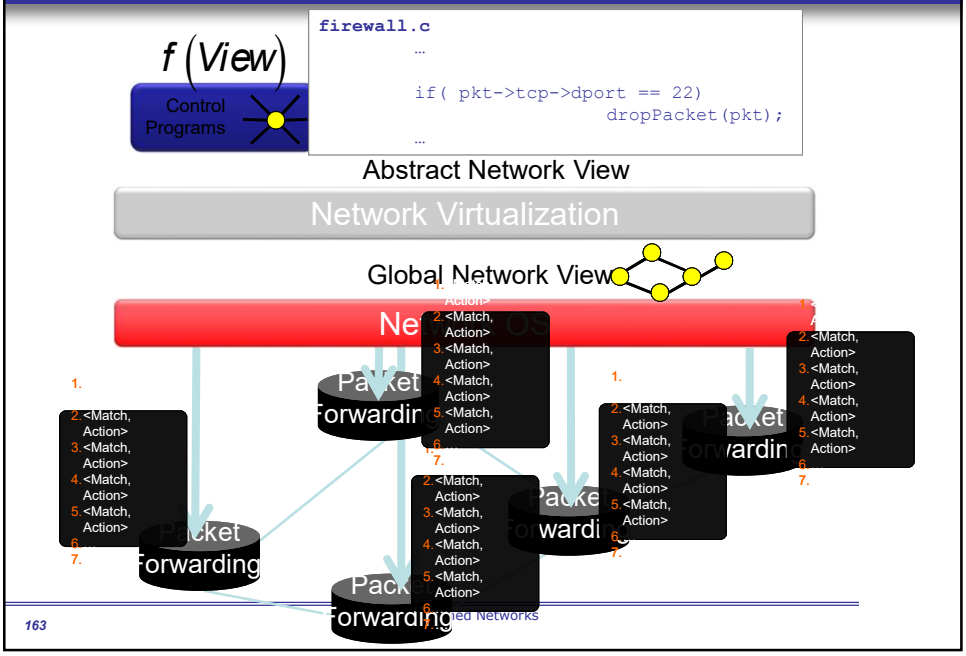
With SDN we can:

1. Formally verify that our networks are behaving correctly
2. Identify bugs, then systematically track down their root cause.

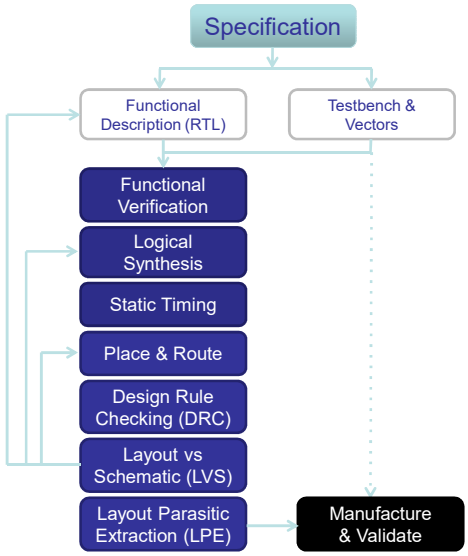
Software Defined Network (SDN)



Software Defined Network (SDN)



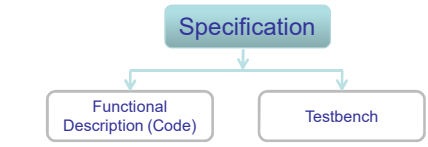
Making ASICs Work



$10B tool business supports a $250B chip industry

100s of Books
>10,000 Papers
10s of Classes

Making Software Work



$10B tool business supports a $300B S/W industry

Static Code Analysis | Invariant Checker | Model Checking

Run-time Checker | Interactive Debugger

100s of Books
>100,000 Papers
10s of Classes

Making Networks Work (Today)

`traceroute, ping, tcpdump, SNMP, Netflow`

…. er, that's about it.

Why debugging networks is hard

Complex interaction

- Between multiple protocols on a switch/router.
- Between state on different switches/routers.

Multiple uncoordinated writers of state.

Operators can't…

- Observe all state.
- Control all state.

"Masters of Complexity"

A handful of books
Almost no papers
~~No classes~~

Philosophy of Making Networks Work

YoYo

"You're On Your Own"

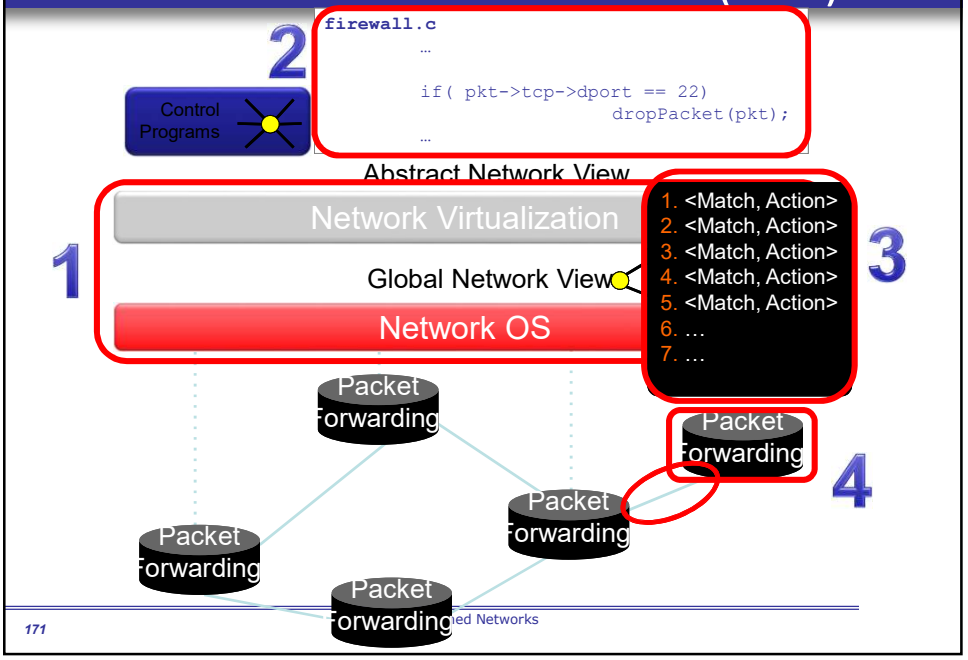
Yo-Yo Ma

"You're On Your Own, Mate"

With SDN we can:

1. Formally verify that our networks are behaving correctly.
2. Identify bugs, then systematically track down their root cause.

Software Defined Network (SDN)



Two SDN projects

1. Static Checking
   "Independently checking correctness"

2. Header Space Analysis
   "Is the datapath behaving correctly?"

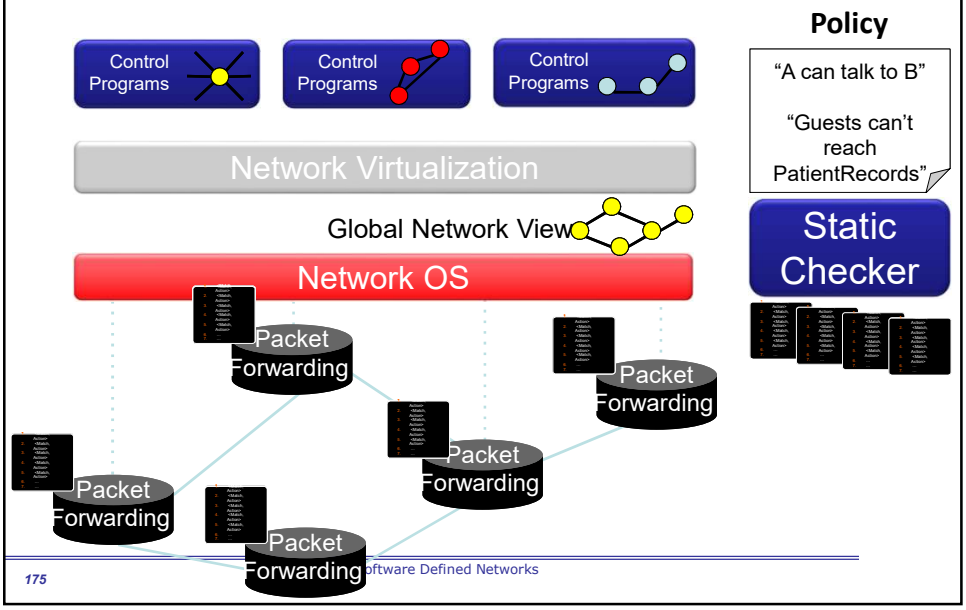
Independently checking correctness

Motivations

In today's networks, simple questions are hard to answer:

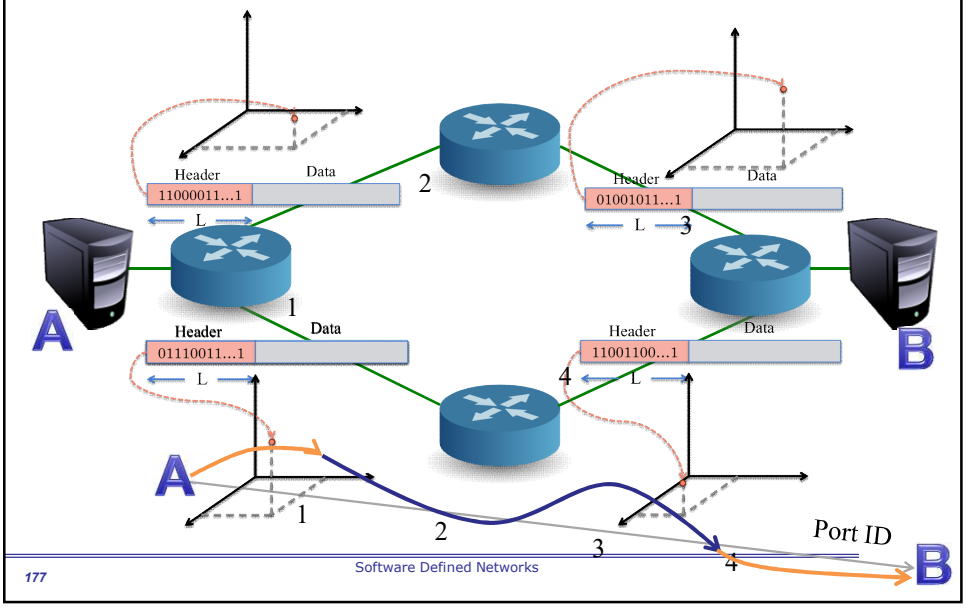
- Can host A talk to host B?
- What are all the packet headers from A that can reach B?
- Are there any loops in the network?
- Is Group X provably isolated from Group Y?
- What happens if I remove a line in the config file?

Software Defined Network (SDN)

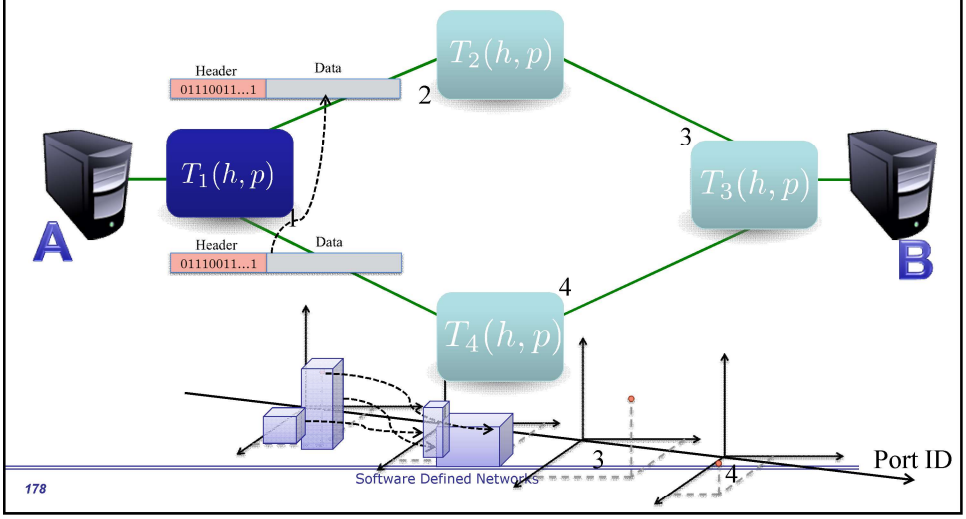


Header Space Analysis

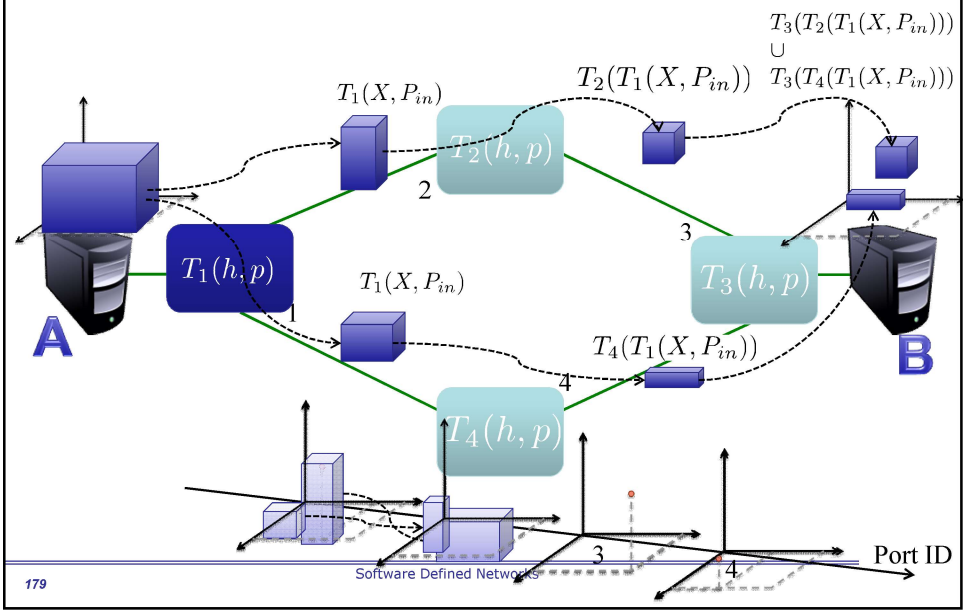
Header Space Analysis



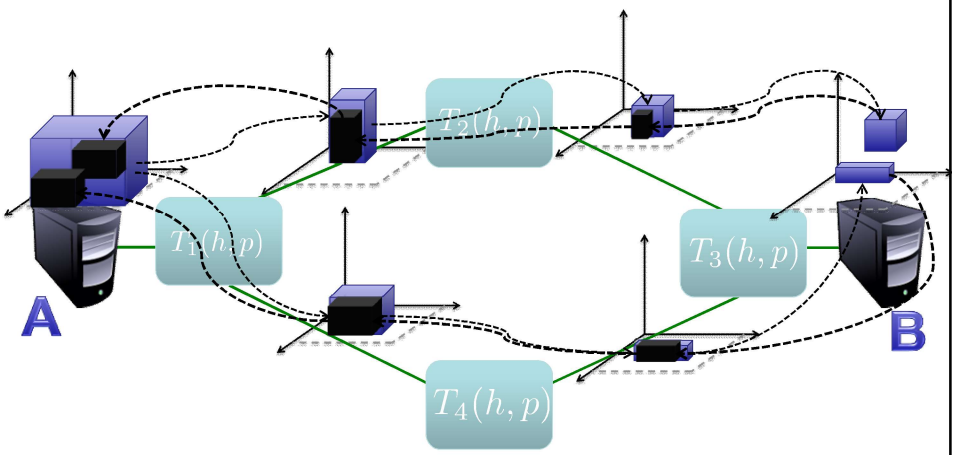
Header Space Analysis



Can A talk to B?



All packets from A that can reach B



Header Space Analysis

Consequences

- Abstract forwarding model; protocol independent
- Finds all packets from A that can reach B
- Find loops, regardless of protocol or layer
- Can prove that two groups are isolated

Can verify if network adheres to policy