Network Virtualization
UNIT-III
Part-I

The material is prepared from these two sources.
Acknowledgement: 1. https://sdn.systemsapproach.org/
2. Lecture notes 'Software Defined Networking Basics', Dr. Anupama Potluri

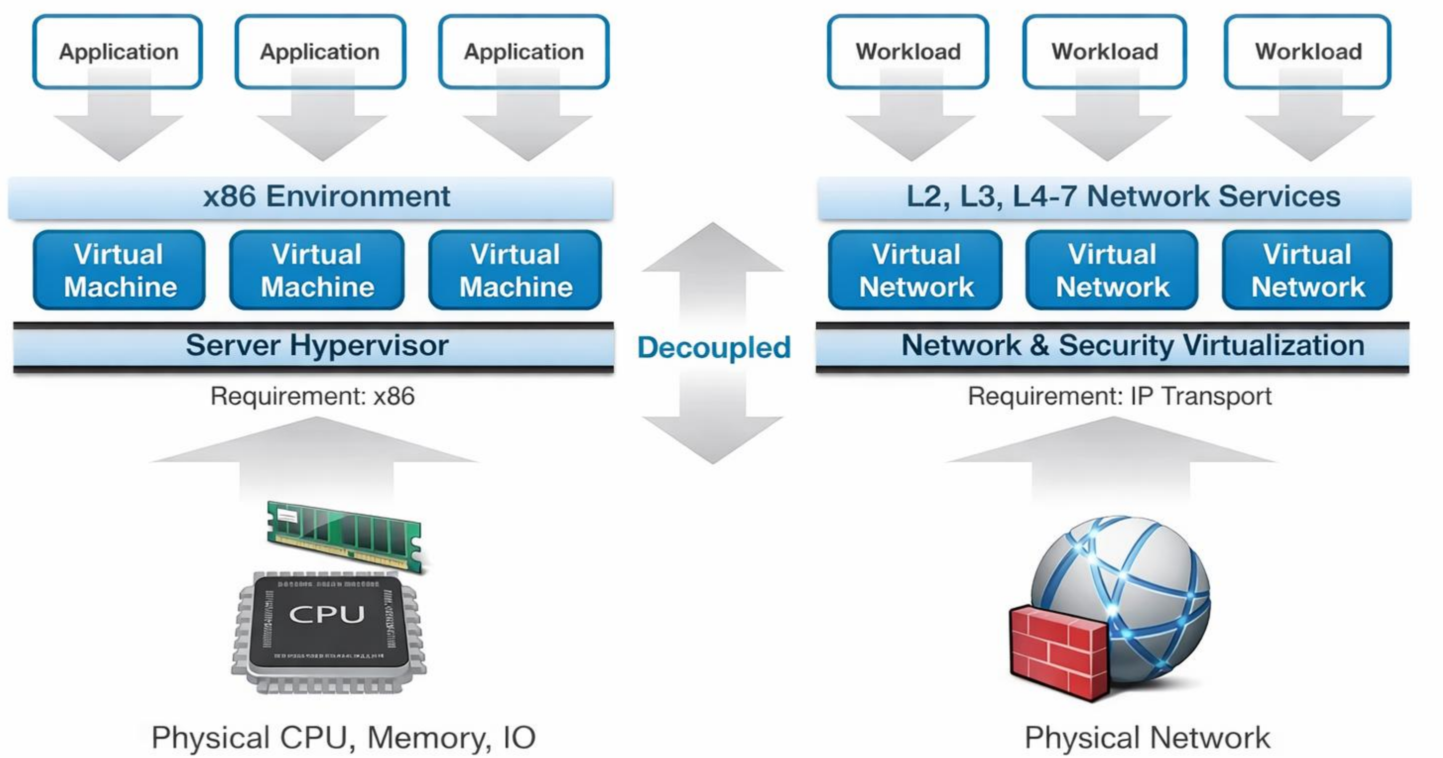
Big question

- We discussed **server virtualization in last two units,** where multiple virtual machines run on the same physical server.
- But once compute was virtualized, a new problem appeared: the **network remained static and hardware dependent**.
- If servers can be virtualized, can networks also be virtualized?

What is Network Virtualization

- In traditional networking:
  - One physical network corresponds to one logical network.
- *Network virtualization is basically an abstraction layer over the physical network*
  - Multiple logical networks on shared physical substrate
  - A Container of network services
- Logical networks appear independent
- Decouples logical topology from physical topology

Network Virtualization

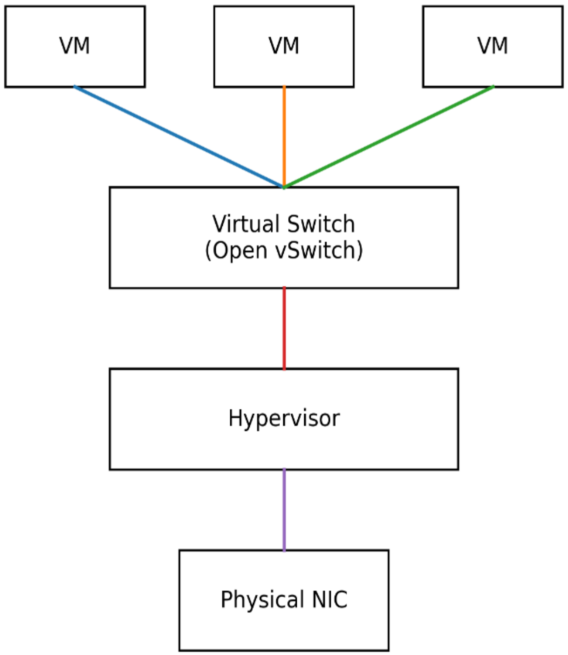


Underlay Network

https://community.cadence.com/cadence_blogs_8/b/breakfast-bytes/posts/how-virtualization-is-changing-networking

First question in the mind

- Is there any hypervisor type software used to do Network Virtualization?
- The network virtualization logic is not done directly by the hypervisor.
- Instead, it is done by a **virtual switch** running inside the hypervisor environment.
- So the hypervisor hosts the virtual switch, but the network virtualization logic is implemented by the virtual switch like vSwitch.



What Can Be Virtualized?

- Network virtualization is not limited to just virtual machines.
- Several components of the network can be virtualized.
- Nodes (Virtual Machines / Containers)
- Links (Tunnels / Encapsulation)
  - Logical links can be created using tunneling mechanisms such as:
    - **V**irtual e**X**tensible **L**ocal **A**rea **N**etwork (VXLAN)
    - **G**eneric **R**outing **E**ncapsulation (GRE)
    - **GE**neric **N**etwork **V**irtualization **E**ncapsulation (GENEVE)
  - These tunnels connect virtual nodes.
- Switches (Virtual switching layer, Ex. Open vSwitch.)

What Can Be Virtualized?

- Address spaces & policies

- This allows different tenants to use the same IP ranges without conflicts.
- Example:
  - Tenant A uses 10.0.0.0/24
  - Tenant B also uses 10.0.0.0/24
- Both can coexist because the networks are isolated.

Why Network Virtualization Emerged

- Compute became elastic; network remained static
  - Cloud required rapid provisioning
  - Manual VLAN provisioning was slow
    - Configuring the network still required:
      - VLAN configuration
      - switch configuration
      - firewall configuration
  - Network became architectural bottleneck

Data Center Evolution

- Enterprise to Cloud to Hyperscale
  - High VM/container density
  - East-West traffic dominance
    - Traditionally most traffic was:
      - North-South traffic (user to server)
    - But in modern clouds most traffic is:
      - East-West traffic (server to server, in modern data centres).
  - Mobility & live migration demands

Multi-Tenancy Requirements

- Shared physical infrastructure
  - Overlapping IP support
    - different tenants may use same private IP range
  - Strong traffic isolation
    - tenants must not see each other's traffic
  - Per-tenant policy enforcement
    - different firewall and routing rules

Limitations of VLAN

- Before network virtualization, VLANs were used to isolate networks
- VLANs have major limitations: 12-bit ID gives us max 4096 networks (Clouds may have thousands of tenants)
  - STP inefficiency (Spanning Tree disables redundant links, which wastes bandwidth)
  - Poor L2 scalability (Data Link Layer)
  - Operational complexity
  - Large broadcast domains increase ARP* storms

*An ARP storm is a severe network congestion event in which an excessive number of Address Resolution Protocol (ARP) requests flood a local area network (LAN), leading to high CPU usage on network devices, significant network slowdowns, or even complete network failure.

Architectural Question

- So network architects started asking several fundamental questions.
  - How to decouple logical from physical?
  - How to scale isolation beyond VLAN limits?
  - How to automate provisioning?
  - How to virtualize network like compute?
- *These questions led to the development of Software Defined Networking and Network Virtualization.*

Why Network Virtualization Was Needed

- Compute virtualization (VMs) had already automated:
  - Server provisioning
  - Storage provisioning
- But network configuration was still manual.
- Virtual machine migration exposed:
  - VLAN limitations
  - Manual bottlenecks
- Network became the “long pole” in cloud deployment *(the slowest step or biggest bottleneck in a process):*

VM Provisioning Bottleneck

- Compute virtualization:
  - Provision of VM in seconds
- Network configuration involves:
  - VLAN setup
  - Firewall rules
  - Routing updates
- This mismatch between compute speed and network speed created a major bottleneck.
- This triggered need for automation.

VM Mobility Problem

- Without virtualization:
  - VM IP tied to physical subnet
  - VM migration breaks connectivity
    - its IP address may become invalid
  - TCP sessions drop
  - Applications restart
- One possible solution was to extend layer 2 networks across the entire data center.
    - But this creates huge broadcast domains and does not scale.
- Network virtualization solves this problem.

Disaggregating the Control and Data Planes

- When separating the control plane and data plane
  - Control plane would decides how packets should be forwarded.
  - Data plane will actually forwards the packets.
- Traditionally both ran inside the same router or switch, but::
  - *Software Defined Networking separates these two planes.*
- Disaggregation in SDN is the separation of the control plane from the data plane so that they can be implemented and managed independently.

Why Separation Was Necessary?

- **Historical Context**
- Traditional networking devices tightly coupled:
  - Control logic
  - Forwarding hardware
  - OS… this vertical stack created vendor lock-in
- Problems:
  - Any innovation required hardware upgrades
  - Closed ecosystems
  - Slow standardization cycles
  - Limited experimentation
- SDN introduced architectural separation to break this rigidity.

Conceptual Separation

- In any network device, two fundamentally different functions exist:
- **Control Plane**
  - Computes network-wide policies
  - Runs distributed algorithms
  - Maintains topology information
  - Responds to failures
- Examples:
  - Border Gateway Protocol (BGP)
  - Open Shortest Path First (OSPF)
  - Routing Information Protocol (RIP)

Conceptual Separation

- In any network device, two fundamentally different functions exist:
- **Data Plane**
- The data plane is responsible for actual packet forwarding.
  - Performs per-packet forwarding
  - Operates at line rate (Tbps)
  - Requires deterministic latency
- When a packet arrives at a switch or router, the device must quickly decide: Which port should this packet leave from?

Conceptual Separation

- Key Insight:
  - – Control plane operates in milliseconds.
  - – Data plane operates in nanoseconds.
- This difference justifies separation.
- So separating control and data planes allows:
  - flexible decision-making in software
  - ultra-fast packet forwarding in hardware.

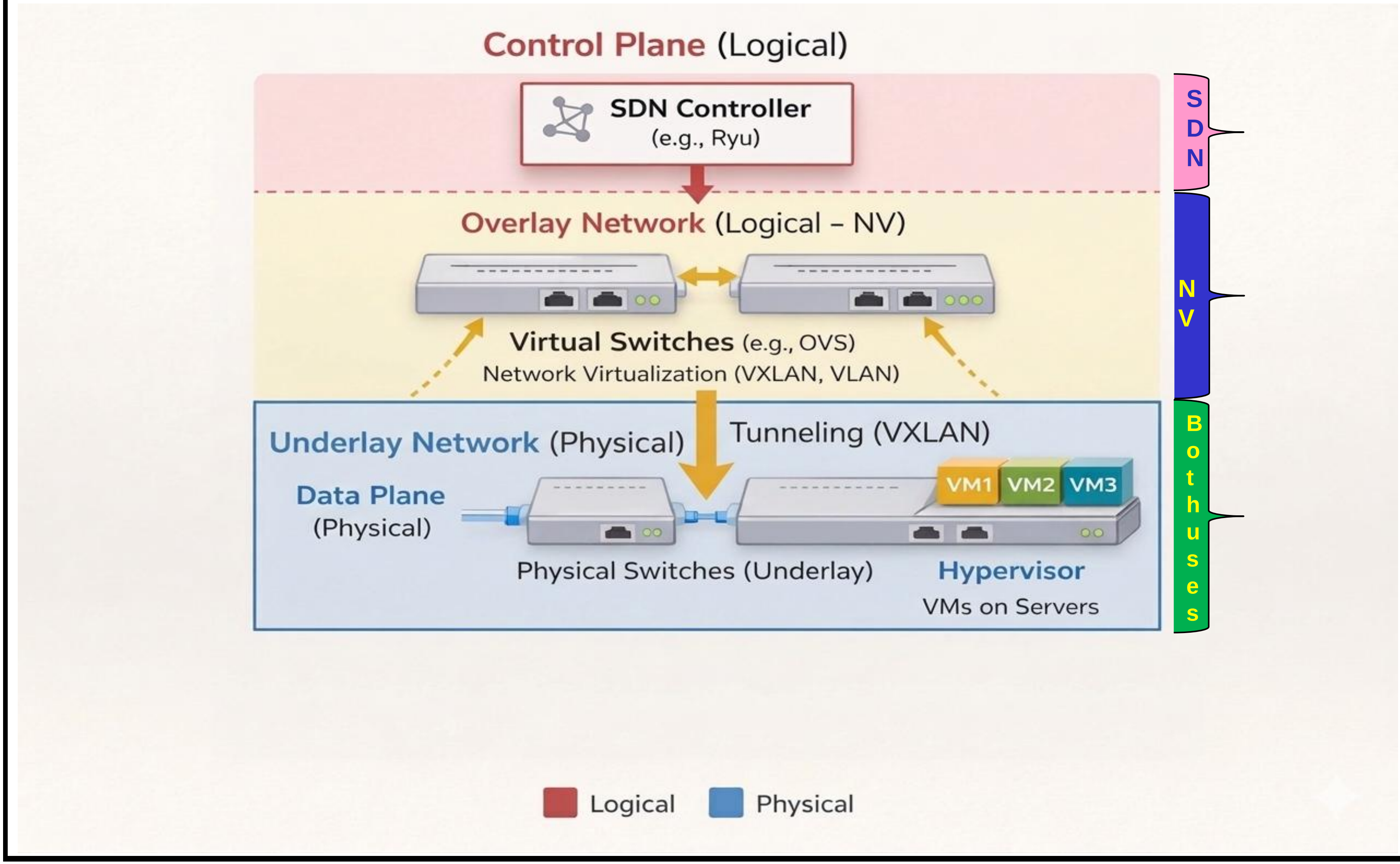
SDN vs NV

- “…it's important to recognize a distinction that::
  - SDN itself does not inherently abstract the details of physical network,
  - essentially what it does is separate the data plane from the control plane but:
- Network virtualization is the technology that provides the abstraction so that multiple logical networks can be instantiated on top of that single physical network.”

SDN vs NV

- "... SDN effectively separates the data plane and the control plane
- whereas virtual networks separate logical and physical networks
- So, SDNs are in some sense a useful tool for implementing virtual networks but the two concepts are certainly distinct from one another."

Integration of SDN with NV



RIB vs FIB: Architectural Implications

- To understand control and data plane separation more clearly, we must understand two **key data structures**:
- **Routing Information Base (RIB)**
  - maintained by control plane
  - Maintains multiple candidate paths
  - Stores metrics, policies, preferences
  - Optimized for computation
- **Forwarding Information Base (FIB)**
  - Derived from RIB
  - Optimized for lookup performance
  - Uses hardware-friendly structures
  - used by the data plane.

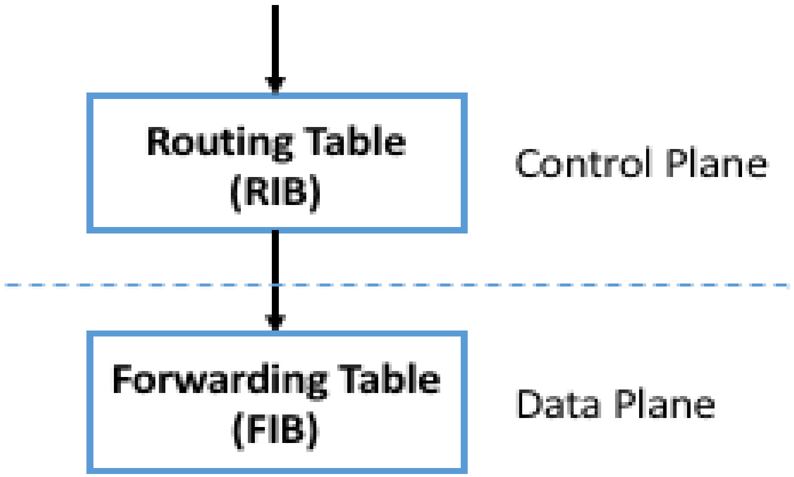
RIB vs FIB: Architectural Implications

- Separation allows:
  - Algorithmic flexibility in RIB

    &

  - Hardware efficiency in FIB

  - RIB = thinking
  - FIB = forwarding



RIB (tentative)

Destination	Next Hop	Protocol	Metric	Selected
10.0.0.0/24	10.1.1.2	OSPF	20	Yes
10.0.0.0/24	10.2.2.2	BGP	200	No
172.16.0.0/16	Direct	Connected	0	Yes
192.168.1.0/24	192.168.1.1	RIP	2	Yes

What Disaggregation Really Means

- Disaggregation is not just separation, it implies:
  - Open interface between planes
  - Independent evolution
  - Vendor-neutral interoperability
  - This encourages innovation and competition.
  - Operators could theoretically buy:
    - Control plane from Vendor A
    - Switch hardware from Vendor B.
- Market competition at each layer

Impact of Disaggregation

Before	After
Monolithic mainframes	Modular PC architecture
Integrated network OS	Disaggregated SDN

- This led to:
  - Development of Bare-metal* switches
    - high-performance packet forwarding devices
    - without proprietary control software.
  - Software-defined control logic

- *A **bare-metal switch** is a **network switch that is sold without proprietary control software**. It usually includes only: Switching hardware, **A**pplication **S**pecific **I**ntegrated **C**ircuit (ASIC) forwarding chip, Ports and interfaces, but **no vendor-specific network operating system**.

Match–Action Abstraction

- A key idea introduced by SDN is the **match–action abstraction**.
- Networking operations can be simplified into conditional logic.
- Structure:
- If packet matches certain conditions then perform specific action.

Match–Action Abstraction

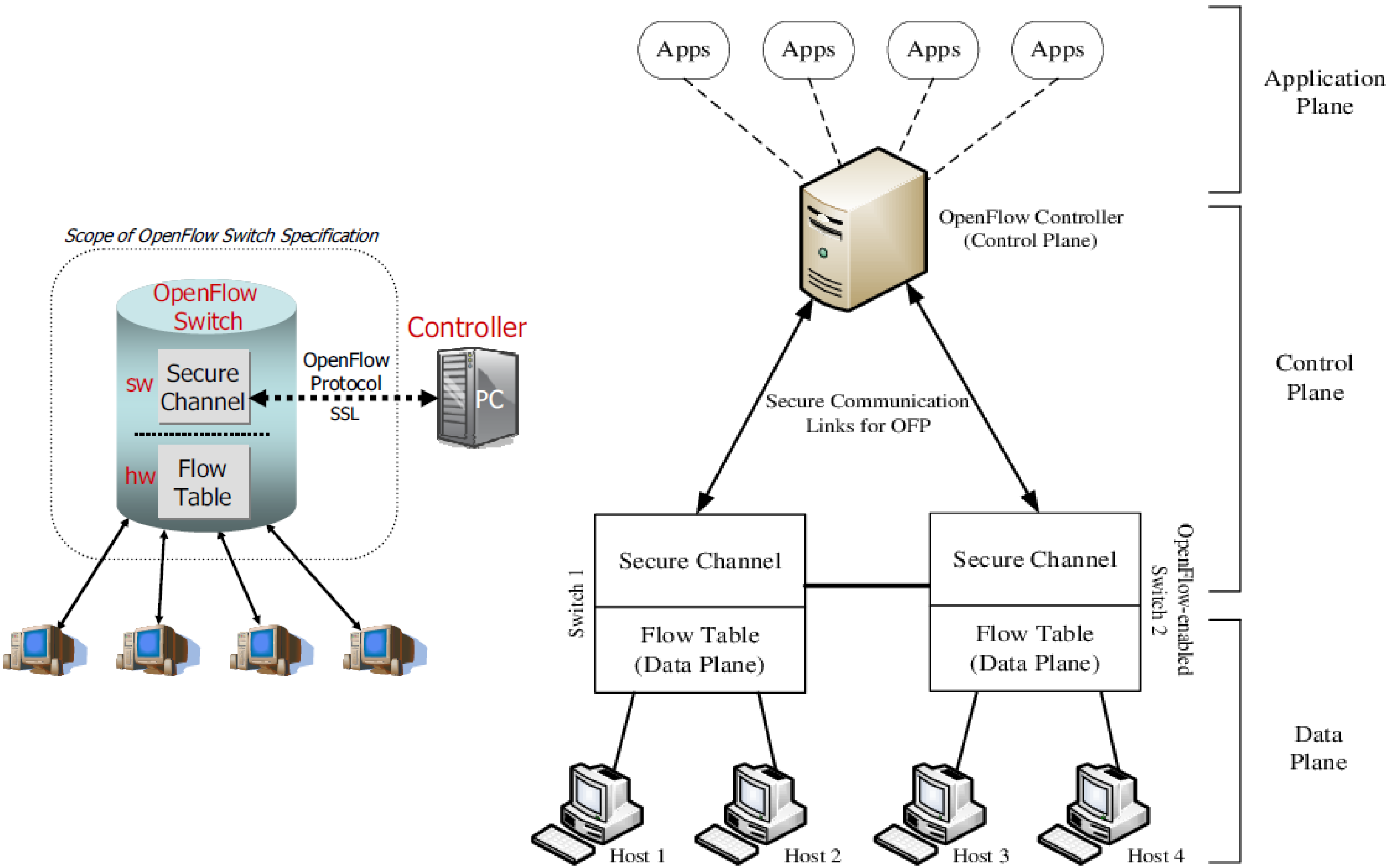
- Match:
  - IP dst = 192.12.69.0/24
  - TCP port = 80
- Action:
  - Forward to Port 3
  - Rewrite MAC
  - Count bytes
- This abstraction is:
  - Simple
  - General
  - Hardware implementable

OpenFlow

- *OpenFlow was one of the first practical implementations of SDN.*
- **OpenFlow** architecture enables Software-Defined Networking (SDN) by separating the control plane from the data plane.
- It consists of three main components:
  - a centralized controller,
  - OpenFlow-enabled forwarding devices (switches), and
  - a secure communication channel connecting them.
- The controller dictates traffic flow by managing flow tables within switches via protocols, allowing for flexible, programmable network management.

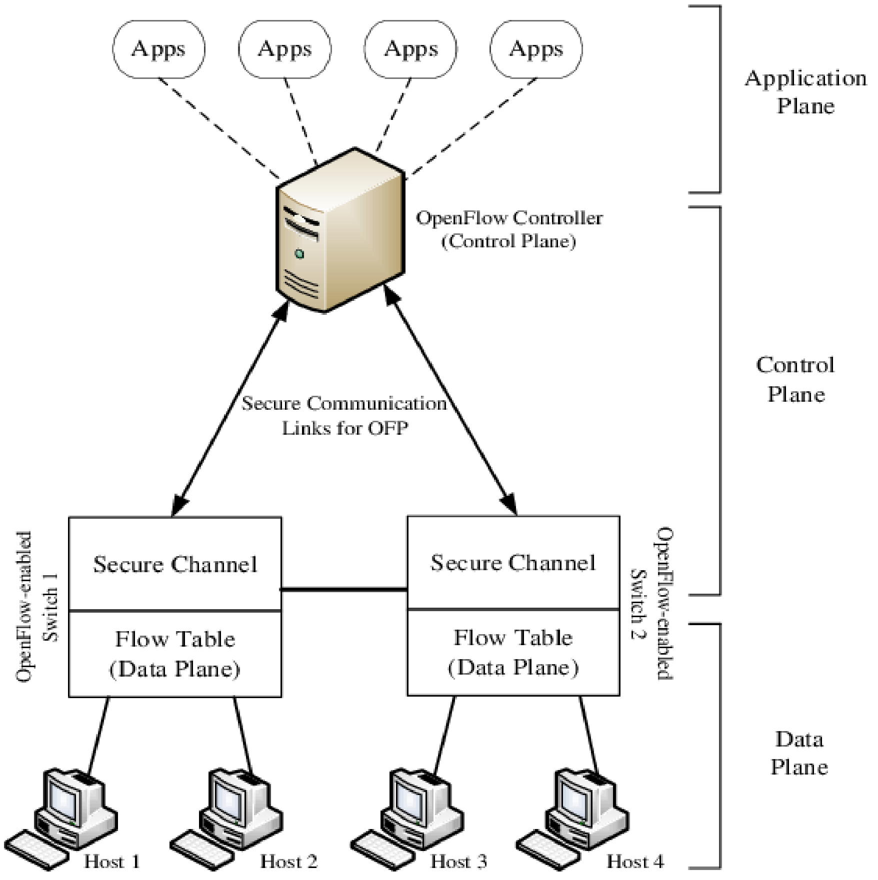
https://collaborate.princeton.edu/en/publications/openflow-enabling-innovation-in-campus-networks/

OpenFlow architectural diagram



Core Components of OpenFlow Architecture

- **OpenFlow Controller (Control Plane)** The **"brain"** of the network, responsible for making forwarding decisions, updating flow tables, and managing network topology.
- **OpenFlow Switch (Data Plane):** Hardware or virtual switches that handle packet forwarding based on entries in their flow tables.
- **Secure Channel:** A secure, usually TLS or TCP-encrypted connection (typically port 6653 or 6633) that connects the switches to the controller.



OpenFlow Concept

- Flow-table driven switches
  - Controller installs rules
  - Match to Action abstraction
    - If a packet matches certain header fields then perform a specified action
  - Event-driven interaction

Proactive vs Reactive Flow Installation

- There are **two ways** to install flow rules.

1. **Proactive Mode**
   - Flow rules installed in advance.
   - No controller interaction during forwarding.
   - Lower latency.
   - Higher memory usage.

- **Trade-Off**
- Latency vs Switch Memory vs Controller Load.

Proactive vs Reactive Flow Installation

- **Idea:**
- Rules are **installed in advance (before traffic arrives)**.
- **How it works:**
  1. The controller analyzes network policy/topology.
  2. It **pushes flow rules to switches beforehand**.
  3. When packets arrive then switch **already knows what to do** so no need to contact controller.
- **Example:**
  – Suppose a network admin knows that: All traffic from **10.0.0.0/24 to 20.0.0.0/24** should go via path A.

Proactive vs Reactive Flow Installation

- The controller installs this rule in all relevant switches ***before any packet is sent***.
- When a packet arrives:
  - Switch matches flow entry it forwards immediately.
- **Advantages:**
  - Very **low latency** (no controller involvement per packet)
  - Suitable for **predictable traffic**
  - Scales better under heavy load
- **Disadvantages:**
  - Needs **pre-planning**
  - Less flexible for dynamic/unexpected traffic

Proactive vs Reactive Flow Installation

2. Reactive Mode

- First packet sent to controller.
- Controller computes policy.
- Installs flow rule in switches.
- Higher latency for first packet.
- Efficient memory usage.

- *Reactive mode introduces **initial latency**.*

Proactive vs Reactive Flow Installation

- **Idea:**
- Rules are **installed on-demand (after traffic arrives)**.
- **How it works:**
  1. A packet arrives at switch.
  2. No matching flow so switch sends **Packet-In** to controller.
  3. Controller decides path.
  4. Controller sends **Flow-Mod** to install rule.
  5. Packet is forwarded.

Proactive vs Reactive Flow Installation

- **Example:**
  - Host A sends packet to Host B for the first time:
  - Switch doesn't have rule so sends packet to controller.
  - Controller computes path & installs flow rule.
  - Subsequent packets follow installed rule.
- **Advantages:**
  - **Highly flexible**
  - Good for **dynamic networks**
  - No need to predefine flows
- **Disadvantages:**
  - **Higher latency** for first packet
  - Controller can become **bottleneck**

Limitations of OpenFlow

- Although OpenFlow was revolutionary, it also had limitations.
- **Flow table scaling**
  - Large networks require millions of flow entries.
- **Ternary Content Addressable Memory (TCAM) constraints**
  - Flow tables are stored in **TCAM memory**, which is:
    - expensive
    - limited
    - power hungry.
- *This limited the scalability of OpenFlow-based systems.*

What then?

- Now we have discussed some of the **limitations of OpenFlow**, such as flow table scaling, TCAM constraints, and controller bottlenecks.
- *If OpenFlow had these limitations, why was it still so influential in networking research?*
- The reason is that OpenFlow was **not just a protocol**. It introduced a set of **fundamental abstractions** that changed how we think about networking.
- Instead of configuring each router individually, SDN introduced the idea of treating the network as a **programmable system**.

Three Fundamental SDN Abstractions

- *SDN introduced three major abstractions that simplify networking.*

- Distributed State Abstraction
  - Controller abstracts distributed network state. Applications do not worry about distributed topology.
- Forwarding Abstraction
  - Generalized match–action data plane.
- Specification Abstraction
  - Control programs express what network should do, not how to implement it.
- These abstractions enable:
  - Modularity
  - Programmability
  - Rapid innovation

Distributed State as an Abstraction

- In traditional networking, each router independently builds its view of the network.
- Using protocols like:
  - Border Gateway Protocol (BGP)
  - Open Shortest Path First (OSPF)
  - Routing Information Protocol (RIP)
- This requires complex distributed algorithms.
- SDN approach:
  - Controller gathers global network information.
  - Maintains **centralized topology view**.
  - Applications simply operate on this global view.

Network Operating System (NOS)

- A Network Operating System is the platform that runs the SDN controller.
- Its job is to:
  - Maintain global network state
  - Provides programmable interface to the network.
  - Hides vendor-specific switch implementation details.
  - Compiles high-level policies into low-level flow rules.

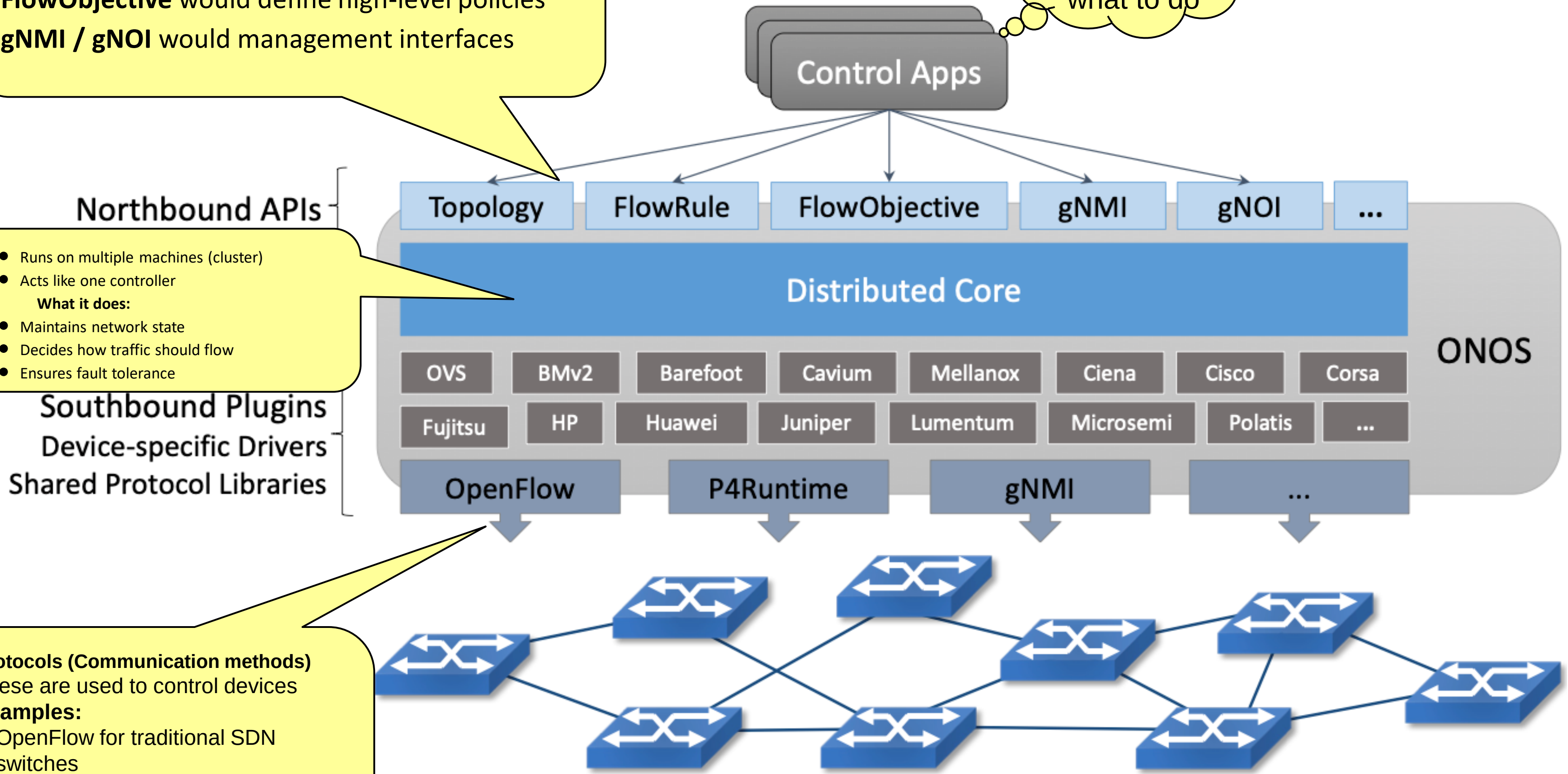
Network Operating System (NOS)

- Acts as middleware between:
  - Control Applications
  - Physical Switches
- Examples:
  - NOX
  - ONOS
  - OpenDaylight

ONOS Architecture

Topology would get network structure
FlowRule would install rules
FlowObjective would define high-level policies
gNMI / gNOI would management interfaces

tell the network what to do



- Runs on multiple machines (cluster)
- Acts like one controller

What it does:

- Maintains network state
- Decides how traffic should flow
- Ensures fault tolerance

Protocols (Communication methods)
These are used to control devices
Examples:
- OpenFlow for traditional SDN switches
- P4Runtime for programmable data planes
- gNMI for modern telemetry/config

Control vs Configuration

- *It is important to distinguish between control and configuration.*
- **Configuration:**
  - Set IP address
  - Set port speed
  - Add static route
- Frequency:
  - Thousands per day.
- **Control:**
  - React to congestion
  - Respond to failure
  - Install flow dynamically
- Frequency of these operation:
  - Thousands per second.

SDN focuses on control, not configuration.

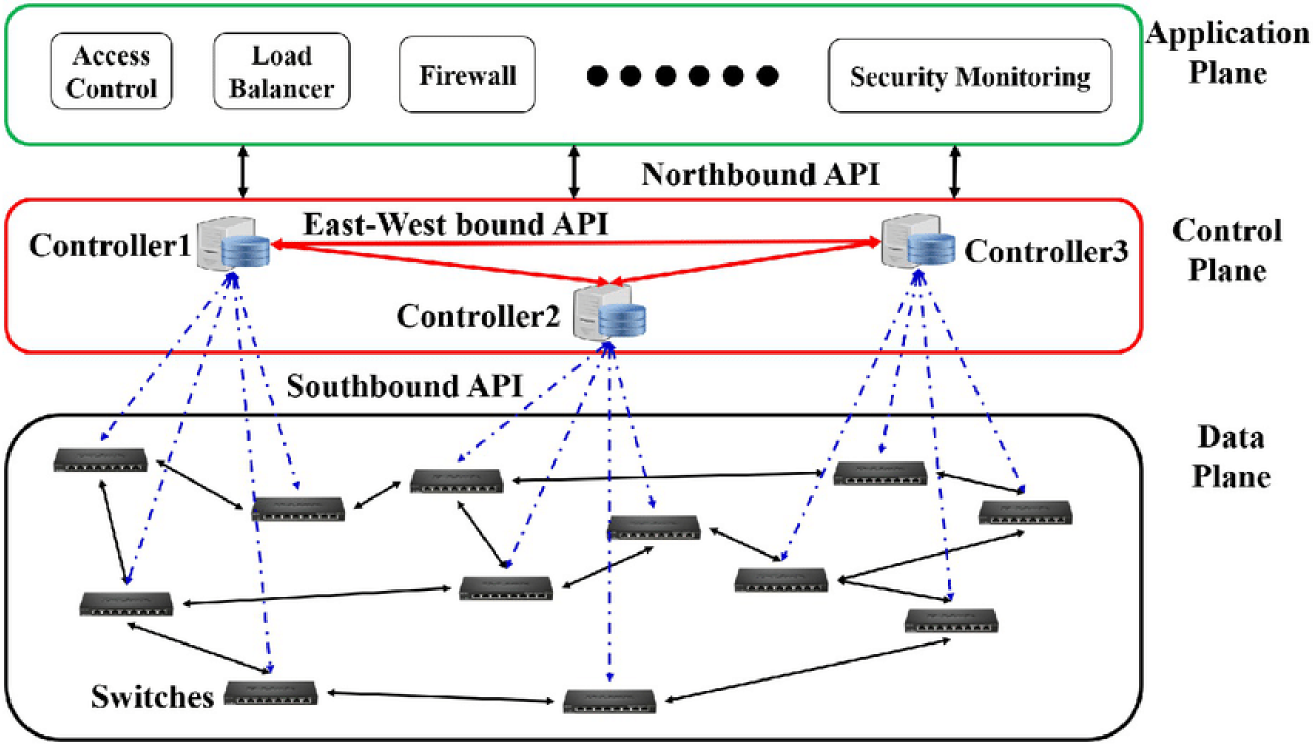
Distributed Control Plane (Traditional Model)

- Each router independently:
  - Maintains its own RIB
  - Exchanges information with neighbors
  - Computes local best path
- Advantages:
  - Proven scalability
  - Resilient
  - Internet-scale deployment
- Limitations:
  - No global view
  - Hard to optimize traffic engineering
  - Convergence delays

Logically Centralized Control

- In SDN:
  - In SDN, the control logic is **logically centralized**.
  - Maintains unified network graph
  - Installs rules in switches
- Implementation:
  - Physically distributed cluster
  - Logically centralized abstraction

Visual Representation of the Definition



Key Idea:

- Controller = Brain
- Switches = Execution units
- Communication via secure southbound interface

SDN Cont..

- **Important clarification**
- From the last figure one can observe that, although we say *centralized*, the controller is usually implemented as a **distributed cluster** for scalability and reliability.
- So it is:
  - physically distributed
  - logically centralized.

Benefits & Risks of Centralization

- Centralized control provides several advantages.
- Global network visibility to controller
  - Simplified policy management
    - Policies are implemented once at the controller instead of configuring each switch.
  - Controller scalability concerns
    - Controller must handle many switches and flows.
  - Failure domain discussion
    - If the controller fails, the network may be affected.

*This is why controllers are implemented as **clusters**.*

Benefits & Risks of Centralization

- Centralization simplifies management but introduces *potential risks*.
- Modern SDN controllers address these risks using:
  - distributed clustering
  - state replication
  - fault tolerance.

TCAM Trade-off

- Switches implement forwarding rules using specialized memory called **T**ernary **C**ontent **A**ddressable **M**emory (TCAM)
- Pros:
  - O(1) lookup
  - TCAM allows Parallel matching
  - high-speed packet processing
- Cons:
  - Expensive, Limited entries, High power consumption
- *This limitation is why SDN controllers must carefully optimize how flow rules are installed.*

Specification Abstraction

- *Network operators describe what they want the network to do, not how to implement it.*
- Network operator specifies:
  - Desired topology
  - Policies
  - Isolation requirements
  - Traffic constraints
- Control program works on **abstract (virtual) topology**.
- Network Virtualization layer maps:
  - Virtual topology to Physical topology.
- Control applications:
  - Do not know physical switch layout.
  - Are unaffected by physical migration or failures.