Advance Computer Networks (CS G525)

BITS Pilani
Pilani Campus

Virendra S Shekhawat
Department of Computer Science and Information Systems

innovate achieve lead

BITS Pilani
Pilani Campus

First Semester 2018-2019
Slide_Deck_M2_2

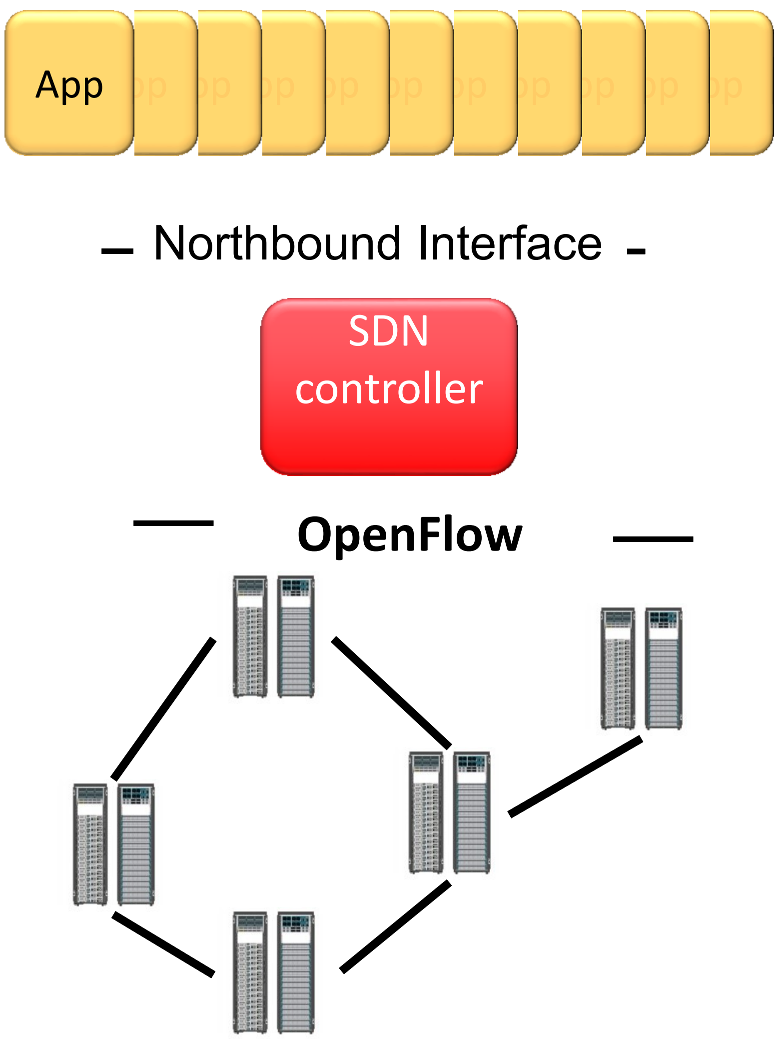
Next…

- SDN Controller
  - aka Network Operating System (NOS)
- Controller's Key Characteristics

SDN Controller Features

- SDN Controller is a software system or a collection of systems
- It provides Management and distribution of network state
  - e.g. topology information, control session information, configuration information
- Gives *High* level data model (Yang modeling language) that captures the relationships between managed sources, policies and other services
- Gives APIs to exposes the controller services to an application

SDN Controllers



- Network elements has two components: **OpenFlow client**, **forwarding hardware** with **flow tables**.
- The SDN controller must implement the network OS functionality
  - Provide **abstraction** to the upper layer
  - Provide **control** to the underlying hardware
  - **Managing** the resources

Ten Key Characteristics of SDN Controller [1]

- OpenFlow Support
- Network Virtualization
  - VLAN (Layer-2) and VRF (Layer-3) are already there ?
  - How server virtualization is different …?
  - Decouple virtual networks from the physical networks
- Network Functionality
  - Routing decision on multiple header fields
  - Multi-tenancy support
  - Support for adding new protocols

Ten Key Characteristics of SDN Controller [2]

- Scalability
  - How Layer-2 networks connect today…?
    - Through Layer-3 functionality. Multiple Layer-3 hops incur delay.
  - How SDN can solve this problem..?
    - Provides a single network view…. Hence provides better scalability
  - How many switches it can support…?
    - Depends on use cases…

Ten Key Characteristics of SDN Controller [3]

- Performance
  - How much time takes to setup a flow?
  - How many flows per second a controller can setup?
  - Flow setup mechanisms:
    - Proactively vs. Reactively
- Network Programmability
  - Different paths for inbound and out bound traffic
  - Dynamically control the traffic based on network conditions change
  - Ability to apply sophisticated filters to packets using multiple packet header fields
  - Enables programmability by implementing north bound APIs

Ten Key Characteristics of SDN Controller [4]

- Reliability
  - SDN controller is a single point of failure
  - Can provide quick setup of fail-over paths
  - Multipath setup
  - Design validation by controller before configuring the network
  - Supports controller Clusters

Ten Key Characteristics of SDN Controller [5]

- Security of the Network
  - Should support authentication and authorization of the network administrators
  - Traffic isolation for each tenant
  - Should provision for network attack detection

- Centralized Monitoring and Visualization
  - Flow level traffic monitoring
  - Global view of network

SDN Controller: NOX/POX

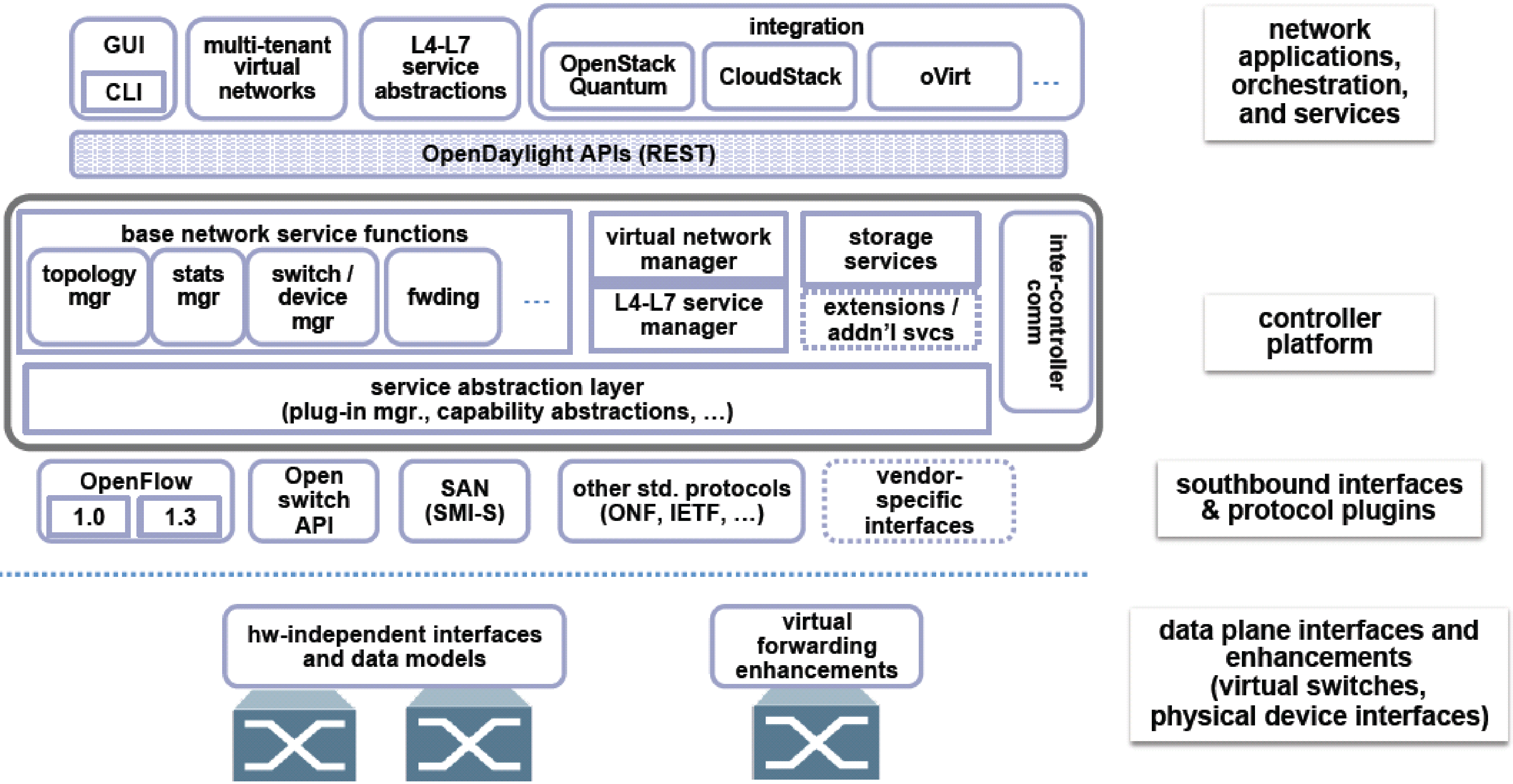
- Originally developed by Nicira
- NOX: C++ version; POX: python version
  - Nox for performance; Pox for rapid prototyping.
- POX comes with Mininet – the simulation infrastructure
- OpenFlow v.1.0
- Programming model:
  - Controller registers for events (**PacketIn**, **ConnectionUP**, etc).
  - Programmer write event handler

NOX/POX Component

NOX/POX controller

- Connection Manager
- Event dispatcher
- OpenFlow Manager
- DSO Deployer
- Input/output Socket Asynchronous File
- OpenFlow API
- Threading and event management
- Other utilities

OpenDaylight Architectural Framework

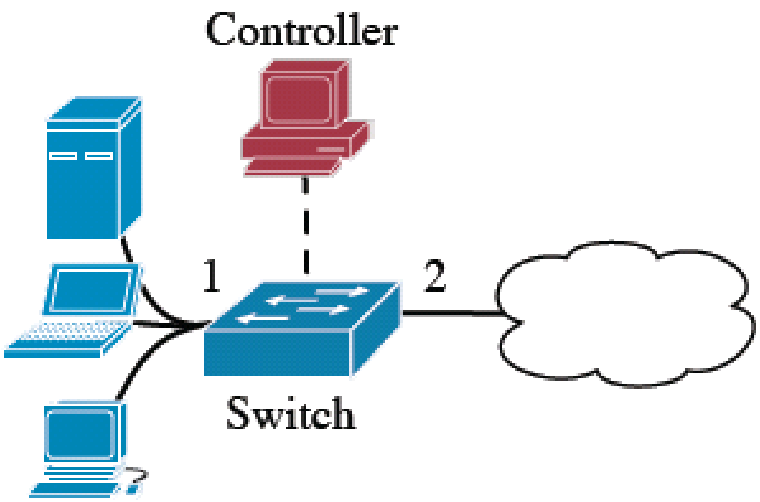


Reference: https://wiki.opendaylight.org

Programming SDN with NOX/POX

- Event driven programming paradigm
  - Application writer register the events and write the event handlers.
- API
  - Events:
    - packet_in(switch, port, packet)
    - stats_in (switch, xid, pattern, packets, bytes)
    - flow_removed (switch, pattern, packets, bytes)
    - switch_join(switch)
    - Port_change(switch, port, up)
  - Control and query openflow switches
    - Install(switch, pattern, priority, timeout, actions)
    - uninstall(switch, pattern)
    - Query_stats(switch, pattern)

Programming SDN with POX: A Repeater Application



- Network app: when the switch is up, install rules for repeater (port 1 to port 2, port 2 to port1)

```
def switch_join(switch):
  repeater(switch)
def repeater(switch):
  pat1 = {in_port:1}
  pat2 = {in_port:2}
  install(switch,pat1,DEFAULT,None,[output(2)])
  install(switch,pat2,DEFAULT,None,[output(1)])
```

Programming SDN

- Directly programming over NOX/POX does not have enough abstraction
  - The global network view by itself does not help
  - It is almost like assembly programming
  - Need higher level language for SDN programming
    - C++, python for Net apps.
    - Program at a higher level and then compile into the lower level API.

- Need better abstraction at the language level
  - Like all other high level languages, the abstraction needs to be sufficiently expressive for network applications
  - A good abstraction needs to have some properties that other programming language has (software reuse, etc).
  - We need to be able to implement them efficiently over NOS API.

SDN Network Updates

- Minimum updates within a single switch
- Network-wide Consistent updates

The minimum update problem

	Pattern	Priority	Action
A	[1, 2, *]	5	Port 1
B	[*, 2, 3]	4	Port 2
C	[1, *, 4]	4	Port 3
D	[1, *, 3]	3	Port 4
E	[*, *, 4]	3	Port 5
F	[*, *, 3]	2	Port 6

- Inserting rule G [*, 2, 4] in the table with high priority, above all but below rule A.
  - Insert the rule with the right priority
  - Adjust the priority of other rules if necessary
  - How to set the priority?

The minimum update problem

	Pattern	Priority	Action
A	[1, 2, *]	5	Port 1
B	[*, 2, 3]	4	Port 2
C	[1, *, 4]	4	Port 3
D	[1, *, 3]	3	Port 4
E	[*, *, 4]	3	Port 5
F	[*, *, 3]	2	Port 6

- Rule G: Pattern [*, 2, 4] above all but below Rule A.
  - Rules with overlapping patterns have interdependence
    - Changing priority would change the dependence
  - [*, 2, 4] overlaps with Rule A, C, and E.

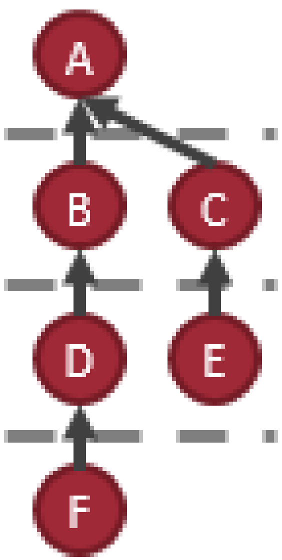
Capturing the dependence in the flow table

- Finding the exact dependence can be tricky:
  - Two rules may be independent even if their patterns overlap.
  - Two rules are directly independent iff the patterns are disjoint or if the joint of the pattern is shadowed by other rules they both depend on.

The minimum update problem

- Represent the minimum dependency in a flow table with a DAG

	Pattern	Priority	Action
A	[1, 2, *]	5	Port 1
B	[*, 2, 3]	4	Port 2
C	[1, *, 4]	4	Port 3
D	[1, *, 3]	3	Port 4
E	[*, *, 4]	3	Port 5
F	[*, *, 3]	2	Port 6



- Insert G [*, 2, 4]   into the DAG

Next…

- Retrospective on Evolving SDN

- Reading
  - Fabric: A Retrospective on Evolving SDN [Martin Casado, 2012]

Background [1]

- Network Infrastructure
  - Hardware
  - Software (Controls the overall behavior of the network)
- Hardware for Ideal Network Design would be:
  - Simple (inexpensive to build and operate)
  - Vendor-neutral
  - Future proof
- Ideal software (control plane) for Network
  - Flexible
    - Supports isolation, virtualization, Traffic engineering, access control and support future requirements

Background [2]

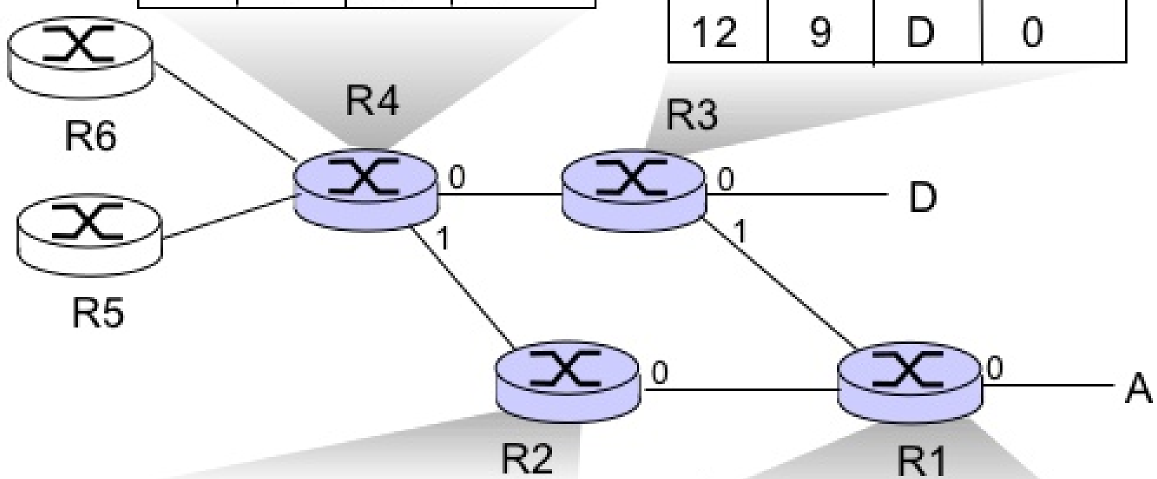
- Question
  - Is today's network (aka conventional network) infrastructure *ideal*….???
  - What are the inadequacies…?

- Observation
  - Inadequacies in these infrastructural aspects are more problematic than Internet's architectural deficiencies

Attempts Made So far…

- Active Networking
  - Flexibility vs. Practicality
- ATM
  - Complicated approach… Why?
- MPLS
  - VPN deployment and Traffic Engineering
- Question
  - Does MPLS meet all goals of an ideal network?

MPLS Forwarding Example



R4:

in label	out label	dest	out interface
	10	A	0
	12	D	0
	8	A	1

R3:

in label	out label	dest	out interface
10	6	A	1
12	9	D	0

R2:

in label	out label	dest	out interface
8	6	A	0

R1:

in label	out label	dest	out interface
6	-	A	0

View of Network

- Network requirements comes from two sources
  - Hosts (aka users) and Operators
- Interface based view of Network
  - ***Host – Network***
    - How hosts inform the network about their requirements?
    - e.g., L3 header tells about the destination address
  - ***Operator – Network***
    - How operators inform the network about their requirements?
    - Usually on per group of packets (manual configuration). How SDN does???
  - ***Packet – Switch***
    - How a packet identifies itself to a switch?
    - e.g. A router use some fields from the packet header as an index to its forwarding table
- Question
  - How these interfaces visualize/realize in the Internet?

Interface view of MPLS

- How MPLS looks like?
  - It does explicit distinction between the **network edge** and the **network core**.
  - Create traffic tunnels between edge router pairs to meet TE requirements and uses labels to forward the traffic in the network core
- Question
  - How these interfaces visualize/realize in MPLS network?
- Answer
  - Host specifies its requirement to the network through IP and packet express its requirement to the switch by MPLS label
    - MPLS distinguishes between *Host – Network* and *Packet – Switch* interfaces
  - *Operator – Network* Interface is missing?

Interface view of SDN

- Provides fully programmatic *Operator – Network* interface
  - Achieves this flexibility by decoupling control plane from the topology of the data plane

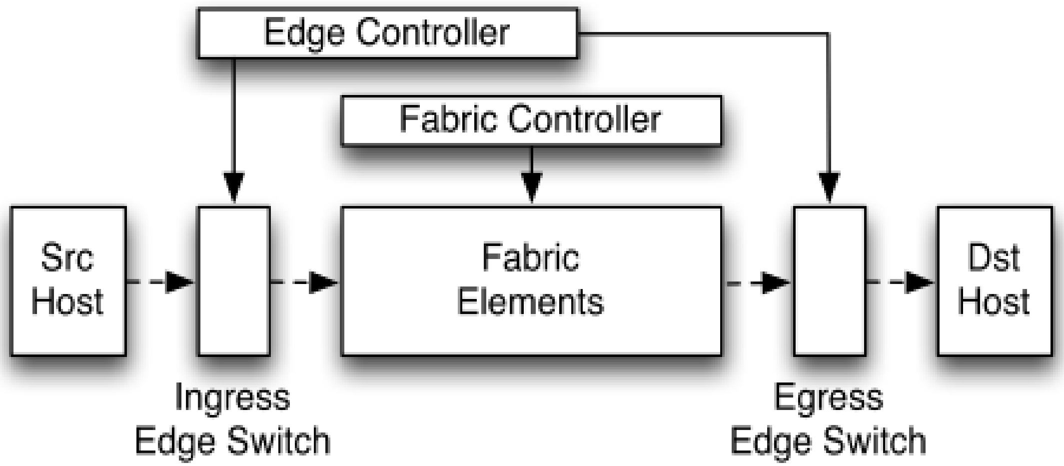
- OpenFlow Switch
  - Exports an interface that allows a remote controller to manage its forwarding state via flow tables

- Question
  - What about *Host– Network* and *Packet – Switch* interface?

Problems with SDN

- OpenFlow switch hardware is not simple
  - Requires to support lookups over so many bits
- Does not provide sufficient flexibility
  - Host requirements to continue to evolve, this leads to increase in generality for *Host-Network* interface
  - This generality must be present on every switch
- It couples the host requirements to the network core behavior
  - Change in external network protocols necessitates a change in the matching behavior (e.g. IPv4 to IPv6)

Extending SDN [1]



- Fabric
  - Collection of forwarding elements, provides basic packet transport
- Edge
  - Responsible for complex network services

Edge and Fabric Separation

- Separation of Forwarding
  - Allows independent evolution of fabric and edge
  - Fabric should provide a minimal set of forwarding primitives without exposing any internal details
  - External address should not used in forwarding decisions

- Separation of Control
  - Two different problems
    - Fabric is responsible for packet transport across the network
    - Edge is supposed to provide more semantically rich services like security, mobility, isolation etc.
  - Separation allows any Fabric can support any edge!!!

Addressing and Forwarding

- Fabric Path Setup
  - Use standard IGP and ECMP
    - Works well in Intra-domain network
  - Use MPLS
    - Works well in Inter-domain network
- Addressing and Forwarding in the Fabric
  - Fabric forwarding elements are differ from traditional forwarding elements… How?
  - Two approaches can be used-
    - Fabric addresses are opaque labels and does MPLS like forwarding (Suitable for both path based or destination based with label-aggregation)
    - Can follow destination based lookup with longest prefix match

Mapping the edge context to the fabric

- When a packet crosses from the edge to the fabric, something in the network must decide with which fabric-internal network address to associate with the packet
  - Two possible ways to do this-
    - Address translation
      - Edge addresses are replaced with fabric internal addresses and translated back into appropriate edge addresses at the destination
      - *Drawback:* Unnecessary coupling between both addresses!
    - Encapsulation
      - Packet is encapsulated with the another header that carries the fabric-internal addresses
      - At destination, this header is removed

Discussions

- Isn't this just another approach to layering?
  - *Edge and Core are now two different Layers*
  - *Horizontal layering*
    - ***Host-Network** interface occurs at the edge*
    - ***Packet-Switch** interface occurs in the core*

- How OpenFlow will be affected?
  - "Edge" and "Core" version of the OpenFlow
  - Edge version can be implemented in the "software" like NFV

Thank You!